AFRL-IF-RS-TR-1999-2
Final Technical Report
January 1999

A PROTOTYPE TWO-LEVEL MULTICOMPUTER STUDY

Myricom, Inc.

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. B861

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-1999-2 has been reviewed and is approved for publication.

APPROVED: Raymond A. Liuzzi

RAYMOND A. LIUZZI
Project Engineer

FOR THE DIRECTOR: Fowler

NORTHRUP FOWLER, III, Technical Advisor
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

A PROTOTYPE TWO-LEVEL MULTICOMPUTER STUDY

Charles L. Seitz

Contractor:	Myricom, Inc.
Contract Number:	F30602-94-C-0270
Effective Date of Contract:	24 September 1994
Contract Expiration Date:	30 December 1996
Short Title of Work:	A Prototype Two-Level Multicomputer
Period of Work Covered:	Sep 94 - Dec 96
Principal Investigator:	Charles L. Seitz
Phone:	(626) 821-5555
AFRL Project Engineer:	Raymond A. Liuzzi
Phone:	(315) 330-3577

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Raymond A. Liuzzi, AFRL/IFTB, 525 Brooks Road, Rome, NY 13441-4505.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
January 1999

3. REPORT TYPE AND DATES COVERED
Final Sep 94 - Dec 96

4. TITLE AND SUBTITLE
A PROTOTYPE TWO-LEVEL MULTICOMPUTER STUDY

5. FUNDING NUMBERS
C - F30602-94-C-0270
PE - 62301E
PR - B861
TA - 00
WU - 01

6. AUTHOR(S)
Charles L. Seitz

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Myricom, Inc.
325 N. Santa Anita Ave.
Arcadia CA 91006

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

Air Force Research Laboratory/IFTB
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-1999-2

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Raymond A. Liuzzi/IFTB/(315) 330-3577

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The objectives of this project were the rapid development and the evaluation of a high-performance prototype of a new type of multicomputer that promised many advantages for defense applications. This architecture is called a two-level multicomputer because each node employs a primary processor for message handling and other runtime tasks, and one or more secondary processors for performing the user computation. The specific, anticipated advantages of this node architecture include: low software overhead in message handling, because the primary processor is dedicated to packet handling and other runtime functions, so that the secondary processors can concentrate on the user's computation; high performance per unit cost, power, and size, due to the efficiency of the architecture and the level of integration in its implementation; very rapid hardware and software development of nodes that exploit the latest processor chips as the secondary processor; the ability to support heterogeneous configurations in embedded applications. Technology transition was integrated throughout this project, from the SAN chip set, to the two-level multicomputer, to its programming systems, to its applications. This includes: The UC Berkeley NOW, a cluster of SPARC workstations with Myrinet and the "hotbot" (www.hotbot.com) search engine produced by Inktomi Corp, a commercial spinoff of the Berkeley NOW project, and NSWC Dahlgren for distributed computing for AEGIS ships using actual engagement codes.

14. SUBJECT TERMS
Computers, Architecture, Hardware/Software, Information Processing

15. NUMBER OF PAGES
156

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

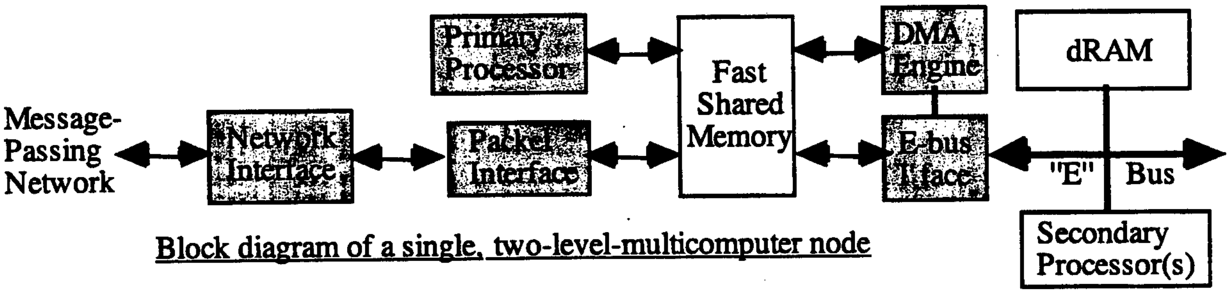
Contents

In somewhat the same style as links in World Wide Web pages, the Executive Summary refers to the Appendices for technical details. Current versions of some of these Appendices are also maintained as Web pages on the Myricom Web site, http://www.myri.com/, and are so noted if this is the case.

Executive Summary

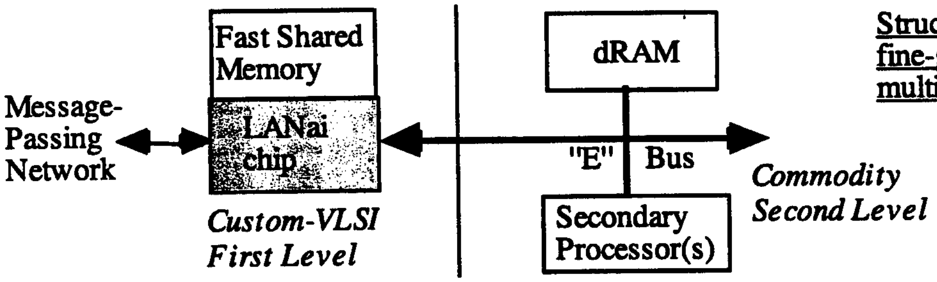
1. Background and Objectives

The objectives of this highly focused, 27-month, research-and-development project were the rapid development and the evaluation of a high-performance prototype of a new type of multicomputer that promised many advantages for defense applications. This architecture is called a *two-level multicomputer* because each node employs a primary processor for message handling and other runtime tasks, and one or more secondary processors for performing the user computation.



Block diagram of a single, two-level-multicomputer node

All of the shaded blocks in the diagram above are contained within a single custom-VLSI chip, a microarchitecture called the "LANai" (see section 2.1 below). Structurally, then, the node is simpler than the block diagram above:



Structural picture of the fine-grain, two-level-multicomputer node

The specific, anticipated advantages of this node architecture include:

- low software overhead in message handling, because the primary processor is dedicated to packet handling and other runtime functions, so that the secondary processors can concentrate on the user's computation;

- high performance per unit cost, power, and size, due to the efficiency of the architecture and the level of integration in its implementation;

- very rapid hardware and software development of nodes that exploit the latest processor chips as the secondary processor, inasmuch as most of the hardware components and system software in the primary-processor part of the node is the same for different secondary processors; and

- the ability to support heterogeneous configurations in embedded applications, including configurations that support high-data-rate network connectivity to sensors and to other systems.

2. Approach and Results

The two-level-multicomputer architecture was conceived and studied by the same group of researchers in earlier DARPA projects at Caltech and Myricom. Although the architecture promised many advantages over previous multicomputers, it needed to be evaluated by designing and constructing a prototype, including its programming system, and by applying the prototype to applications representative of defense missions.

2.1 The SAN Chipset

The two-level multicomputer employs commodity processor, memory, and logic chips together with three custom-VLSI chips -- the System-Area Network (SAN) chipset -- whose design, fabrication, and characterization were the major focus of the first 15 months of this effort.

- *The LANai network-interface chip,* which includes a 32-bit RISC, and its associated fast SRAM form the first level of the multicomputer nodes. The original plan called for two versions of the LANai chip, one with a 32-bit E bus (the LANai 4), and one with a 64-bit E bus (the LANai 5).

- *High-degree, cut-through, switch chips* (called XBar chips) form the multicomputer message-passing network.

- *A Myrinet-interface chip* (called MI chips) with integrated low-voltage differential signaling line drivers and receivers allows long-distance links and connectivity to Myrinet LANs.

The target performance for the SAN or LAN links was for them to carry packet data at a full-duplex rate of 1.28+1.28 Gigabits/second.

The SAN chipset design, fabrication, and characterization were all completely successful, including achieving all functional capabilities and performance targets. The design, fabrication, and characterization of the LANai 4, 8-port XBar (XBar8), and MI chips were all completed by December 1995, 15 months into the contract. The LANai 5 chip was designed and was in fabrication at the completion of the contract (chips received in January 1997), and was characterized under a subsequent DARPA contract.

The LANai chips are advanced descendants of the nodes of the "Mosaic" multicomputer that our research group designed and built at Caltech. The LANai includes all of the mechanisms pioneered in the Mosaic for streamlined message handling.

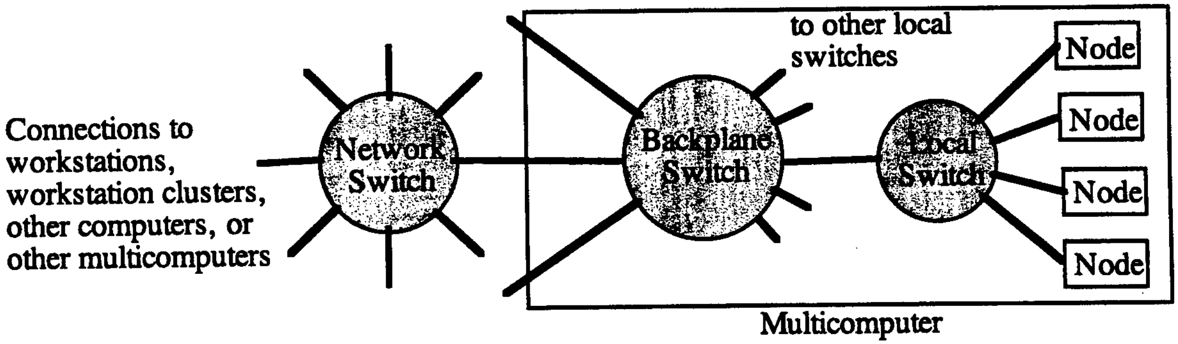
Appendix A is the documentation of the LANai 4 chip.
Appendix B is the documentation of the LANai 5 chip.

Current versions of these documents are linked from

http://www.myri.com/vlsi/

2.2 VME Packaging and the Message-Passing Network

The prototype multicomputer components are packaged on 6U VME boards that are inserted into standard VME subracks. This was not the form of packaging originally proposed, but was adopted upon the suggestions of the DARPA program managers in order to assure compatibility with prototypes produced under other DARPA and DoD projects.



In the diagram above, multiple nodes may be connected by the "Local Switch" on a single VME board. For flexibility in the prototype, some of the SAN message-passing-network ports for a VME board are on controlled-impedance connectors on the VME front panel. These ports are connected through microstrip cables to ports of SAN switches, which are also packaged in the VME form factor.

However, it was important to find an alternative to the front-panel cables, which interfere with board swapping, and would be disallowed in, for example, VME subracks in submarines. Myricom joined with CSPI, a company with experience in supplying VME signal-processing equipment for defense platforms, in developing and characterizing the first prototypes of backplane-overlay switches that employ a new, extra, "P0" connector in VME systems.

Appendix C is a slide presentation illustrating this packaging scheme.

As also indicated in the diagram above, individual VME subracks can in turn be connected to each other and to workstations or other computers through Myrinet "network switches" in a Myrinet local-area network.

In spite of our initial skepticism about the use of VME packaging, **this "Myrinet-on-VME" packaging developed in the project was extremely successful, being widely adopted by other DARPA projects, and now commercially.** In fact, "Myrinet on VME" became a draft standard (VITA-26) developed by the VITA Standards Organization (VSO) of the VME International Trade Association, and is now an American National Standard (ANSI/VITA 26-1998, Myrinet on VME Protocol Specification). The draft standard is linked from both http://www.myri.com/open-specs/ and http://www.vita.com.

Appendix D, chapter 7, includes a detailed description and photographs of the VME packaging used in the prototype two-level-multicomputer hardware.

2.3 Types of Prototype Nodes

Myricom's original plan for this research proposed a selection procedure for developing, programming, and demonstrating different types of two-level-multicomputer nodes with different commodity processors for the second level, including, for example:

- RISC nodes
- floating-point DSP nodes
- integer DSP nodes

The structure of the two-level-multicomputer node and its software allows a newly introduced processor chip to be designed quickly into a new node type.

For the hardware, the first level is implemented simply with a LANai chip and SRAM, and needs only to be interfaced to the bus of the second-level processor. For the software, the system software and network protocols are standardized in the first-level processor. Each type of secondary processor requires only a compiler and a small library.

One of the major "revelations" of this research came about from following the thread of the argument above to its limits. Because the secondary processors do not require a runtime or operating system, they can be simple, low-cost, low-power, performance-oriented devices that do not even need to be able to execute a program in the conventional sense. They could, for example, be vector arithmetic units, without a "CPU." The most interesting technological possibility is that *the secondary processors could be field-programmable gate arrays (FPGAs) that are dynamically configurable from the first-level processors.* Thus, several months into this project, we added

- dynamically configurable FPGA nodes

to the list of node types.

One way to summarize the main *hypothesis* of this research, albeit somewhat flippantly, is that the Principal Investigator offered this "challenge" at a number of DARPA meetings: "Give us a new processor with a C compiler, and in four months we'll be able to show you this processor running user programs in a two-level multicomputer."

The hypothesis of this research was proven even more convincingly, because not only did Myricom develop two-level-multicomputer nodes, but also engineering groups at other companies and research organizations.

RISC nodes. Myricom developed, programmed, and demonstrated RISC nodes more easily than was expected when this work was proposed. The introduction of VME single-board computers (SBCs) that accepted PCI interfaces in the "PCI Mezzanine Card" (PMC) form factor was identified by the Sandia ATR research team as their preferred approach; hence, we developed a PMC version of Myricom's commercial Myrinet-SAN/PCI interfaces together with VME-form-factor Myrinet-SAN switches, both for our own use and for Sandia's use. The SBCs used in these early experiments were Motorola MVME-1602 PowerPC boards, the same as those that were being used by Sandia.

By the time we were ready to develop a more highly integrated VME board with multiple RISCs (e.g., PowerPCs) on each VME board, CSPI had committed to developing this board as a product. We learned of CSPI's product plans under a non-disclosure agreement, but with CSPI's permission we reported the substance of these plans to our DARPA and Rome Labs contract management. We all agreed that our developing such a prototype would be redundant, and that our efforts would thus be better spent on the more innovative FPGA nodes.

The end results of the RISC-node efforts are also described in section 3 on technology transition.

Floating-point DSP nodes. As soon as this contract was awarded, the Myricom research team was introduced by the DARPA management to Bob Graybill and members of his research team at Martin Marietta Labs (MML). Bob and his team had proposed to DARPA a specific "High-Performance Scalable Computing" (HPSC) project based on Analog Devices SHARC floating-point DSPs, but employing a much lower performance and less uniform network. When Bob Graybill and his group saw Myricom's plans, they proposed to use our technology, to which Myricom and DARPA agreed. Thus, the MML team (which was moved to Lockheed-Martin Sanders after the Lockheed, Martin-Marietta merger) took on the development of the two-level-multicomputer nodes with floating-point DSP secondary processors.

This Sanders HPSC effort was highly successful, not only in developing working, high-performance, quad-SHARC nodes that interoperated correctly over Myrinet with other two-level-multicomputer prototypes and with workstations, but in their system software developments, MPI middleware, and prototype application programs.

The successes of the HPSC efforts demonstrated that the Myricom and Sanders teams worked well together. A close collaboration was necessary, inasmuch as the HPSC modules were developed to employ LANai 4 and XBar8 chips that were being designed and characterized concurrently with the HPSC board designs and software.

Other end results of this Myricom-Sanders effort are described in section 3 on technology transition.

Integer DSP nodes. The inclusion of integer DSP nodes in our original planning was stimulated in part by the announcement of the TI C80, a DSP chip that promised ~1 Gop/s performance. Some preliminary designs of a C80-based two-level-multicomputer node were done, but our evaluation of the C80 software and an assessment of there being no technology-roadmap future for the C80 persuaded us and the DARPA management that this effort was not justified.

Dynamically reconfigurable FPGA nodes. The developed two-level-multicomputer hardware and software infrastructures made the design of FPGA nodes and their integration into the programming environment relatively straightforward. The research issues were the choice of an appropriate application, and the programming (generating the configuration) of the FPGAs.

We recognized that FPGA nodes had the potential of offering uniquely high performance per cost, power, and size, but *only over a narrow range of applications*. The silicon-area and performance penalties for logic to be field-programmable are such that FPGAs cannot compete with RISCs or DSPs by being programmed to be RISCs or DSPs. In fact, we conjecture that the application span over which FPGAs can provide higher performance than custom-silicon RISCs and DSPs are computations offering high degrees of concurrency, and in which the data types are those that are not native to RISCs and DSPs (e.g., not just integer and floating-point operands and results).

Other than their limited application span, the other disadvantage of dynamically reconfigurable FPGAs as computing devices is that, at least with present tools, the "programming" of an FPGA is perhaps one to two orders of magnitude more difficult and time-consuming than it would be to express the same computation as a sequential program.

We expect from our own experience that the application-span and programming-difficulty disadvantages of FPGAs as computing devices will limit their use as computing devices in the commercial world. For certain computations, however, they seem to offer advantages in cost, power, and size that would make them attractive for defense applications.

What are these applications? In order to "prove the point," we selected as a test and demonstration application an Automatic Target Recognition (ATR) algorithm developed by a DARPA research project at Sandia National Laboratory. This application and Myricom's implementation of this computation are described in section 2.5 below, and in detail in Appendix D.

2.4 Programming System

The programming system developed for the two-level multicomputer is based on Objective C. Earlier efforts based on C++ demonstrated that concurrent extensions of object-oriented programming systems are highly effective; however, Objective C was selected over C++ due to much easier compiler development and maintenance.

Myricom completed the first software distribution of its Objective-C based Lyric programming system, and in February 1996 hosted the first meeting of the Myricom Multicomputer User's Group. This meeting was aimed at stimulating the development of application programs by other research groups. The meeting attracted forty attendees from DARPA, Hughes, Lockheed Sanders, Sandia, USAF Rome Labs, Mississippi State University, Caltech, USC, and USC/ISI. Myricom researchers presented a project overview and the Lyric programming system, and Tony Skjellum of MSU presented the MPI extensions of Lyric. Most of the material presented at the meeting is available at:

http://www.myri.com/research/darpa/multicomputer_users/

When the two-level multicomputer acts as a network server, application modules written in Objective C can be invoked through the network remote-procedure-call (RPC) mechanism employed by many high-level and application-specific programming systems, or by using the functions of the Message Passing Interface (MPI) standard. In this connection, project researchers contributed to the development of the IETF "PacketWay" standard for interoperability between high-performance packet networks, both those within a multicomputer and those that connect heterogeneous networks of computers.

2.5 ATR Challenge Application

As noted above, Myricom researchers selected as a test and demonstration application an Automatic Target Recognition (ATR) algorithm developed by a DARPA research project at Sandia National Laboratory. The defense mission driving this research is the synthetic-aperture radar (SAR) surveillance performed by the Joint STARS. In this mission, the radar produces SAR images of a very large number of objects distributed across a thousands of square miles. The ATR task is to identify those images that would be of military significance, such as a tank, a fuel truck, or a missile launcher.

The kernel ATR computation following sensor processing, "focus of attention" (FOA), and second-level detection, is a binary image correlation of the SAR image against a database of objects. This database would, for example, include the templates for each of several types of tanks, and the templates for each tank would be represented at any orientation within several-degree increments. Because there are so many SAR images and so many templates, the computation is highly concurrent. This mission can benefit from a nearly open-ended computational capacity.

The Sandia ATR research team set up their computing problem as a "defense challenge application" by publishing the basic algorithms, and by providing sanitized versions of a set of templates. The performance of ordinary platforms, such as a PowerPC running a reference, C-program version of the algorithm, are known, and are expressed in templates per second per node (TSN). The Myricom FPGA-node implementation achieved 1316 TSN by benchmark measurements, compared with ~750 TSN for a (contemporary) 200MHz PowerPC. The actual "bottom-line" metric is templates per second per VME slot, and, because there are 4 FPGA nodes per VME slot in this prototype implementation, but only one PowerPC per slot in a contemporary SBC, the

FPGA node implementation exhibited higher performance per VME slot by a factor of ~7.

As noted in the report in

> Appendix D, "Scalable, Network-Connected, Reconfigurable, Hardware Accelerators for an Automatic-Target-Recognition Application"

there are many other algorithmic optimizations that could improve the performance to ~2500 TPN with the same hardware.

Although the performance of both RISCs and FPGAs continues to escalate, we expect that, when comparing hardware of the same year, the FPGA-node approach to the binary-image correlation will continue to exhibit about an order of magnitude higher performance per VME slot, and there appear to be many other computations that could benefit from this approach.

3. Technology Transition

Technology transition was integrated throughout this project, from the SAN chip set, to the two-level multicomputer, to its programming systems, to its applications. We list here with some pride the **impact** that Myricom's two-level-multicomputer project has had on defense, research, and commercial computing.

Myrinet Products. Myricom itself uses the SAN chipset in its Myrinet products, which are widely used in cluster- and distributed-computing projects and products. For example:

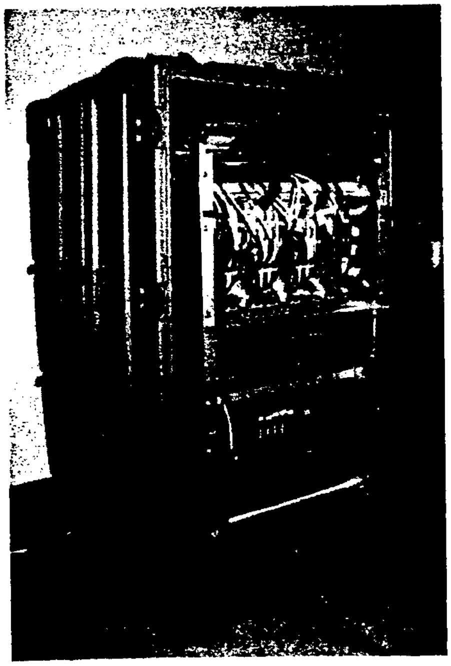
- The UC Berkeley NOW, a DARPA project, is a cluster of SPARC workstations with Myrinet. This cluster has achieved sub-10μs latencies between UNIX processes under their Active Messages implementation, and holds both of the world records for sorting.

- The "hotbot" (www.hotbot.com) search engine produced by Inktomi Corporation, a commercial spinoff of the Berkeley NOW project, has been reported by several independent benchmarks to be the world's fastest Web-search engine. Hotbot is a Myrinet cluster of SPARC workstations.

- NSWC Dahlgren has been evaluating Myrinet for distributed computing for AEGIS ships using actual engagement codes. In earlier tests, Myrinet exhibited the highest data rates and lowest latencies of all of the LANs tested.

- Several DoE ASCI (Accelerated Strategic Computing Initiative) projects employ large Myrinet clusters in support of the ASCI nuclear stewardship mission. (The largest of these clusters today consists of 400 DEC-Alpha hosts.)

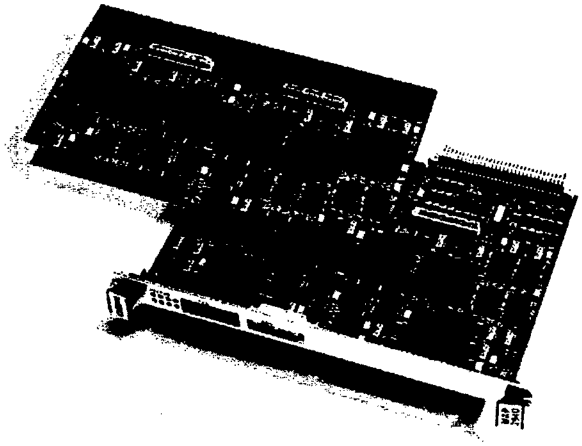
- The system-level integration of the ATR testbed at Sandia National Laboratory is a Myrinet that connects components in both the SAN and LAN forms. The following is a photograph of the Sandia ATR testbed in its shipping crate, ready to be sent to fly on the Joint STARS. The upper VME subrack carries PowerPC SBCs connected by Myrinet-SAN cables (which have red tags because they are carrying classified data). The lower VME subrack carries several Lockheed Sanders HPSC SHARC boards. This system is connected by Myrinet-LAN links to external workstations.



The SAN chipset. Myricom supplies SAN-chipset chips, either packaged and tested or as bare die, to other DARPA contractors as required. Lockheed Sanders uses these chips both in the VME HPSC boards used in the system above, and also in modules that are planned for several insertion projects. Projects at the Universities of Utah and Virginia are designing specialized Myrinet interfaces that employ Myricom chips. Myricom is also supplying board-level and chip-level multicomputer components to Hughes (Raytheon).

The two-level-multicomputer architecture. Myricom worked actively with several VME-processor manufacturers to use the two-level-multicomputer architecture and components to provide scalable, high-performance interconnect for VME systems. These VME companies are suppliers to military-system companies and to the services of high-speed computing and signal processing systems.

The first of these VME companies to develop and ship two-level-multicomputer products was CSPI. The following is a photograph of a CSPI quad-PowerPC two-level-multicomputer VME board, introduced in early 1997 (shortly after the completion of this contract), the first commercial two-level-multicomputer product.



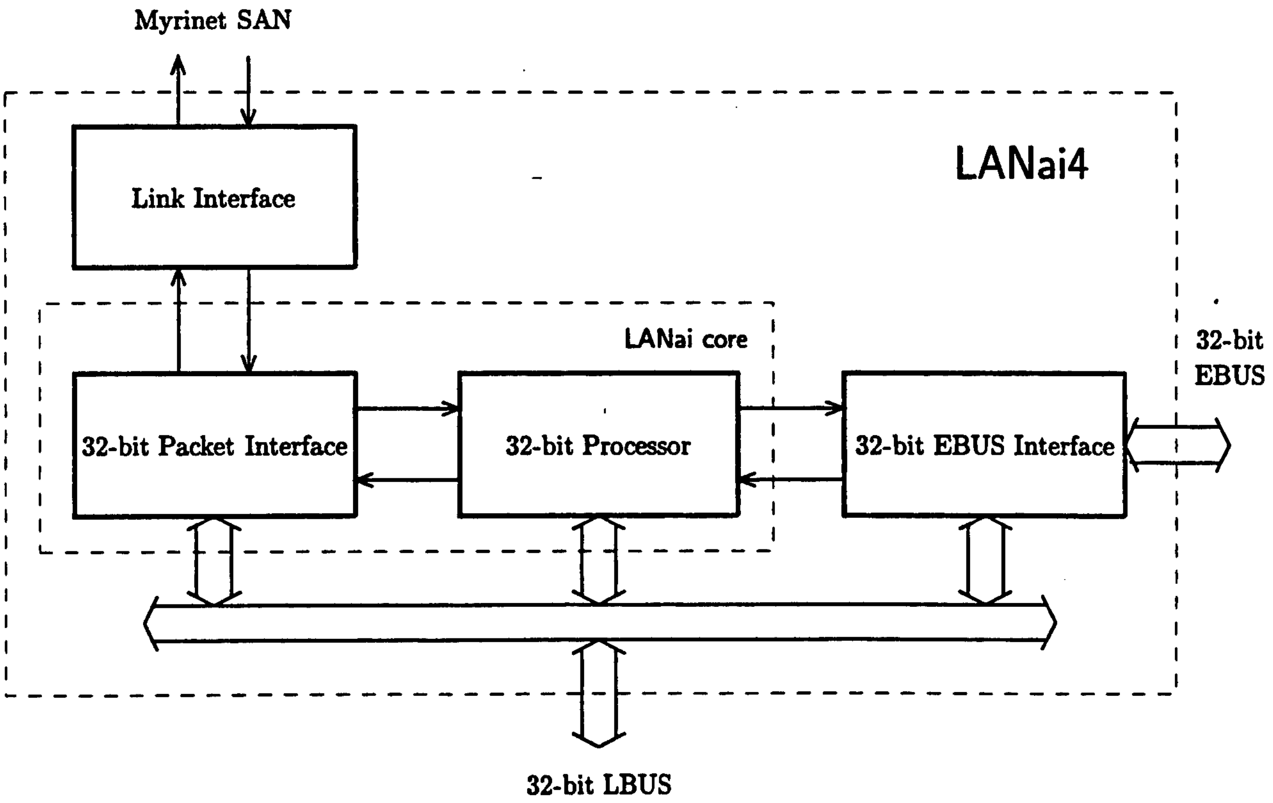
These CSPI products interoperate fully with Myricom's Myrinet products, and the programming system for these CSPI signal-processing boards was patterned closely after the Myricom software.

Thus, the two-level-multicomputer technology, architecture, and software was transitioned almost immediately into commercial practice and COTS availability.

The LANai 4 is a programmable communication device that provides an interface to the Myrinet system-area network (SAN).

As illustrated below, a LANai 4 chip consists of the LANai core, with an instruction-interpreting processor and a packet interface, the Myrinet-SAN interface, and the EBUS interface.

The Local Bus (LBUS) is an interface to asynchronous static SRAMs. The External Bus (EBUS) is a synchronous interface.



MEMORY INTERFACE

In the remainder of this specification, we shall refer to 8-bit data units as bytes, to 16-bit units as half-words, and to 32-bit units as words. Although the internals of the LANai 4 chip support 32-bit addresses, pin-count limitations restrict the LBUS to a maximum of 1M bytes.

The LBUS operates at twice the chip-clock speed — there are two memory cycles for every clock cycle. The external-access bus (EBUS), the packet-interface receive DMA, and the packet-interface send DMA each request a maximum of one memory access per clock cycle. The on-chip processor requests up to two memory accesses per clock cycle (instruction and data). The available memory cycles within each clock cycle are assigned based on the following priority (highest to lowest): EBUS, receive DMA, send DMA, and the processor. Since every EBUS memory request is granted, the LANai 4 chip along with the memory on its LBUS appears as a block of synchronous memory when observed from the EBUS.

Both the LBUS and the EBUS addresses are byte addresses, and the byte order is big-endian (the most-significant byte is stored at the lowest byte address).

The word and half-word memory accesses on the LBUS must be aligned; any least-significant bits of an address that would make a memory access non-aligned are ignored.

The LANai chip provides a rudimentary memory-protection mechanism that allows a memory segment of programmable size to be write-protected from the LANai core, i.e., writable only from the EBUS (page 6).

Although the LANai core cannot access the EBUS directly, the on-chip processor can initiate a data transfer between the LBUS and the EBUS (page 5). The LANai EBUS interface is simple and generic, and extra hardware is necessary to connect it to any standard bus.

PACKET SENDING

Please consult pages 13 through 15 for a set of send and receive examples.

A data-communication, flow-control unit is called a flit, and consists of eight data bits plus a tail bit. Packets are of arbitrary length (in flits), and the tail bit marks the last flit of every packet. The byte order in the communication network is big-endian, i.e., the most significant byte of a word (or of a half-word) appears first in the network.

After the LANai 4 chip is out of reset, and prior to any Myrinet-network access, the TIMEOUT, MYRINET, and WINDOW special registers (page 7) must be initialized.

Packets are injected into the Myrinet network by accessing the following special registers:

Register	Description
SB	**Send Byte:** The least-significant byte of the value written into SB is appended to the outgoing packet. SB must not be written unless the send_rdy bit of the special register ISR is set.
SH	**Send Half-Word:** The least-significant half-word of the value written into SH is appended to the outgoing packet (high-order byte first). SH must not be written unless the send_rdy bit of the special register ISR is set.
SW	**Send Word:** The word written into SW is appended to the outgoing packet (most-significant byte first). SW must not be written unless the send_rdy bit of the special register ISR is set.
ST	**Send Tail:** Writing ST completes the outgoing packet. If the CRC-8 is not enabled (page 7), the tail flit contains the least-significant byte of the value written into ST. If the CRC-8 is enabled, the tail flit contains the CRC-8 for the packet, xor-ed with the least-significant byte of the value written into ST. ST must not be written unless the send_rdy bit of the special register ISR is set.
SMP	**Send-Message Pointer:** Address of the first word of the send-DMA memory buffer. This register is incremented by 4 by the packet interface as each word is appended to the outgoing packet, and, upon completion, equals SML+4.
SA	**Send-Message Align:** The two least-significant bits of this register specify how many leading flits (0-3) of the contents of the next-specified send-DMA memory buffer should NOT be appended to the outgoing packet. This register may be used to keep the payload portion of the message 4-byte aligned, even when the length of the routing header is not a multiple of 4. Only the first send DMA following a write into this register is affected.
SML	**Send-Message Limit:** Writing this register initiates a send DMA that appends to the outgoing packet, one word at a time, the contents of the memory buffer starting with the word at address SMP (except for the leading bytes, if specified by SA) and ending with the word at address SML.
SMLT	**Send-Message Limit, with the Tail:** The same as SML, and, in addition, completes the outgoing packet by appending a tail flit with its data field equal to the CRC-8 byte for that packet.

Upon completion of a send DMA, the send_int bit of the special register ISR is set (page 10).

Since the send DMA accesses 32 bits at a time, the send memory buffer must be aligned on a word boundary. Hence, the two least-significant bits of SMP and of SML(T) are hard-wired to zero.

SA is a write-only register, and reading it produces an undefined value. Reading any other send register is a valid operation, but one should note that SML and SMLT are stored in the same physical register. If any of the send registers are written during a send DMA, the resulting behavior is undefined.

PACKET RECEIVING

Please consult pages 13 through 15 for a set of send and receive examples.

A data-communication, flow-control unit is called a flit, and consists of eight data bits plus a tail bit. Packets are of arbitrary length (in flits), and the tail bit marks the last flit of every packet. The byte order in the communication network is big-endian, i.e., the most significant byte of a word (or of a half-word) appears first in the network.

After the LANai 4 chip is out of reset, and prior to any Myrinet-network access, the TIMEOUT, MYRINET, and WINDOW special registers (page 7) must be initialized.

An incoming packet is accepted from the network by accessing the following special registers:

Register	Description
RB	**Receive Byte:** Reading RB, an 8-bit special register, consumes one byte off of the incoming packet. RB must not be read unless the byte_rdy bit of the special register ISR is set.
RH	**Receive Half-Word:** Reading RH, a 16-bit special register, consumes one half-word off of the incoming packet (the first byte consumed becomes the most-significant one). RH must not be read unless the half_rdy bit of the special register ISR is set.
RW	**Receive Word:** Reading RW, a 32-bit special register, consumes one word off of the incoming packet (the first byte consumed becomes the most-significant one). RW must not be read unless the word_rdy bit of the special register ISR is set.
RMP	**Receive-Message Pointer:** Address of the first word of the receive-DMA memory buffer. This register is incremented by 4 by the packet interface as each word is written into the buffer. After an entire packet has been received, RMP points to the first aligned word past the end of the packet.
RML	**Receive-Message Limit:** Writing RML enables a receive DMA and instructs the packet interface to put the (remainder of the) incoming packet, one word at a time, into the memory buffer that starts at RMP and ends at RML. When the CRC-8 is enabled (page 7), if the message arrives with the correct CRC-8, zero is written into the last byte of the message.

When an entire incoming packet has been transferred into the receive memory buffer, the recv_int bit of the special register ISR is set (page 10). If the receive memory buffer has been exhausted (the last word written is at the location pointed to by RML, and RMP=RML+8), buff_int bit of ISR is set. After a receive DMA is initiated, one must not initiate another receive DMA until the recv_int bit, the buff_int bit, or both, have been set set.

Since the receive DMA accesses 32 bits at a time, the receive memory buffer must be aligned on a word boundary. Hence, the two least-significant bits of RML and of RMP are hard-wired to zero.

It is possible to read past the tail flit of the incoming packet (with RH, RW, or receive DMA). In such cases, the packet following the currently accessed packet is guaranteed not to be corrupted. The orun_bits of the special register ISR show how many flits past the tail flit have been read (page 10). The bytes corresponding to the flits past the tail flit are undefined.

RB, RH, and RW are not physical storage registers, and writing any of them is a vacuous operation. Reading RMP or RML is a valid operation. If any receive operation is initiated during a receive-DMA transfer, the resulting behavior is undefined.

EBUS-LBUS DATA TRANSFER

In the typical operating regime, a LANai 4 chip operates as a slave device on the EBUS. In this regime, the LANai 4 chip along with the memory on its LBUS appears as a block of synchronous memory when observed from the EBUS.

The LANai 4 chip incorporates a DMA engine that can be instructed to perform data transfer between the LBUS and the EBUS, and, in this regime only, the chip acts as a master on the EBUS. The LANai EBUS interface is simple and generic (page 23), and extra hardware is necessary to connect it to any standard bus.

The EBUS-LBUS DMA engine is controlled by the following 32-bit, special registers:

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>LAR</td><td><b>LBUS Address Register:</b> Points to the beginning of the DMA buffer on the LBUS. This register is incremented by 4 as each word is transferred, and, upon completion, points to the first word past the LBUS DMA buffer.</td></tr>
<tr><td>EAR</td><td><b>EBUS Address Register:</b> Points to the beginning of the DMA buffer on the EBUS. This register is incremented by 4 as each word is transferred, and, upon completion, points to the first word past the EBUS DMA buffer.</td></tr>
<tr><td>DMA_CTR</td><td><b>DMA Counter:</b> Writing a non-zero value into the DMA_CTR register initiates a DMA transfer. This register is decremented by 4 as each word is transferred, and equals 0 upon completion.</td></tr>
<tr><td>DMA_STS</td><td><b>EBUS-LBUS DMA Burst Sizes:</b> The four least-significant bits of this register specify the allowed burst modes of the EBUS interface.
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>3</td><td>Enables 64-byte bursts</td></tr>
<tr><td>2</td><td>Enables 32-byte bursts</td></tr>
<tr><td>1</td><td>Enables 16-byte bursts</td></tr>
<tr><td>0</td><td>Enables 8-byte bursts</td></tr>
</table></td></tr>
<tr><td>DMA_DIR</td><td><b>EBUS-LBUS DMA Direction:</b> The least-significant bit of this register controls the direction of the EBUS-LBUS DMA transfer.
<table>
<tr><th>Bit 0</th><th>DMA Direction</th></tr>
<tr><td>0</td><td>LBUS→EBUS</td></tr>
<tr><td>1</td><td>EBUS→LBUS</td></tr>
</table></td></tr>
</table>

Upon completion of an EBUS-LBUS DMA the dma_int bit of the special register ISR is set (page 10).

Since the EBUS DMAs transfer 32 bits at a time, the LBUS memory buffer must be aligned on a word boundary. Hence, the two least-significant bits of DMA_LAR and of DMA_CTR are hard-wired to zero.

The DMA_STS and DMA_DIR are write-only registers, and reading either of them produces an undefined values. Reading any other register is a valid operation. Writing any of the above registers during an EBUS-LBUS DMA transfer may result in violation of the protocol on the I/O bus that the EBUS connects to.

INTERNET-CHECKSUM COMPUTATION

The LANai 4 chip includes a mechanism to compute a partial Internet checksum. The partial checksum is stored in the special register CKS.

Register	Description
CKS	**Internet-Checksum Register:** This register is modified as a side effect of the EBUS-LBUS DMA transfers. Upon completion of an EBUS-LBUS DMA transfer, the CKS register contains the result of the 32-bit, 1's-complement addition of its initial value and the values of all transferred data items (the DMA engine transfers 32-bit data items only).

A typical 16-bit-Internet-checksum computation consists of: writing zero into CKS; performing one or more EBUS-LBUS DMA transfers; and, adding the most- and least-significant half-word of CKS (in software) using 1's complement addition.

Writing the CKS register during an EBUS-LBUS DMA transfer may corrupt the checksum computation.

COUNTERS/TIMERS

There are two real-time counters on the LANai 4 chip, both of which use the time reference that is equal to 40 times the period of the transmit clock of the Myrinet-SAN interface (page 33). Nominally, this is an 80 MHz clock, so the time reference is equal to 1/2 microsecond.

Register	Description
RTC	**Real-Time Clock:** This is a 32-bit counter that is incremented every time-reference period.
IT	**Interrupt Timer:** This is a 32-bit counter that is decremented every time-reference period. Whenever this counter makes a transition from 0x00000000 to 0xFFFFFFFF, the time_int bit of the special register ISR is set (page 10). Whenever it makes a transition from 0x80000000 to 0x7FFFFFFF, the wdog_int bit of ISR is set.

MEMORY PROTECTION

The LANai 4 chip provides a rudimentary memory-protection mechanism that allows a memory segment of programmable size to be write-protected from the LANai core, i.e., writable only from the EBUS.

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>MP</td><td><b>Memory Protection:</b> If the WE (Write Enable) bit is 1, the LANai is allowed to write to any memory location (no memory protection). Upon reset, this is the default value of the WE bit.
If the WE bit is 0, the A12-A19 bits (the A bits) define the region(s) of memory in which the LANai core is allowed to write: a write to a memory location is allowed if a bit in the address of that memory location is 1 and the corresponding A bit is 1.
<table>
<tr><td><b>Bit</b></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td></tr>
<tr><td><b>Name</b></td><td>WE</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td><b>Bit</b></td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr>
<tr><td><b>Name</b></td><td>-</td><td>-</td><td>-</td><td>-</td><td>A19</td><td>A18</td><td>A17</td><td>A16</td></tr>
<tr><td><b>Bit</b></td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td></tr>
<tr><td><b>Name</b></td><td>A15</td><td>A14</td><td>A13</td><td>A12</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td><b>Bit</b></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td><b>Name</b></td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
</table>
</td></tr>
</table>

A typical use of this mechanism is to write-protect a memory segment at the bottom of the memory (where the code for the LANai processor is usually kept), and allow writes to addresses up to the highest available memory on the LBUS.

For example, writing the value 0x0007E000 to the MP special register write-protects the lowest 8KB and allows writes to addresses up to 512KB-1.

Pin-count limitations restrict the LBUS of the LANai 4 chip to a maximum of 1M bytes.

The MP special register is a write-only register, and reading it produces an undefined value.

CONFIGURATION

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>TIMEOUT</td><td><b>Incoming-Message Blocking Timeout:</b> the two least-significant bits of the TIMEOUT special register specify the timeout period of the LANai watchdog timer. If the LANai 4 chip fails to consume an incoming message from the network for the duration of the timeout period, the watchdog timer sets the nres_int bit of the special register ISR (page 10), and, if the NRES_ENABLE bit of the special register MYRINET is set, it resets the LANai chip.
<table>
<tr><th>Value</th><th>Timeout Period</th></tr>
<tr><td>0</td><td>1/16 second</td></tr>
<tr><td>1</td><td>1/4 second</td></tr>
<tr><td>2</td><td>1 second</td></tr>
<tr><td>3</td><td>4 seconds</td></tr>
</table></td></tr>
<tr><td>WINDOW</td><td><b>SAN-Link Sampling Window:</b> The two least-significant bits of this register select the width of the character-time window for the input section of the Myrinet-SAN interface. After the chip is out of reset and prior to any Myrinet access, the value 3 must be written to this register, and a minimum of 10 milliseconds must be allowed for the SAN link to reconfigure.</td></tr>
<tr><td>MYRINET</td><td><b>Myrinet-Link Configuration:</b> The three least-significant bits of this register enable the error-handling features of the Myrinet-SAN interface. After the chip is out of reset, this register must be written prior to any Myrinet access.
<table>
<tr><th>Bit</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NRES_ENABLE</td><td>When the LANai 5 chip fails to consume an incoming message for the duration of the period selected by the TIMEOUT register, the nres_int bit of the special register ISR is set (page 10). If the NRES_ENABLE is set, the chip will be reset when the nres_int bit is set.</td></tr>
<tr><td>1</td><td>CRC8_ENABLE</td><td>Enables the CRC-8 computation.</td></tr>
</table></td></tr>
<tr><td>DEBUG</td><td><b>Hardware-Debug Register:</b> The four least-significant bits of this register select one of 16 internal signals to be output on the WIN pin, for timing observation.</td></tr>
<tr><td>CLOCK</td><td><b>Internal-Clock Phase-Adjusting Register:</b> This special register controls the on-chip clock generation. During the power-on reset, the system-specific value must be written to this register from the EBUS (page 12). This register may be modified only while the chip is in reset (page 21).</td></tr>
</table>

PROGRAMMABLE OUTPUTS

Register	Description
LED	**LED Register:** Bits 0 of this special register is driven to the LED output pin. Bits 1 through 10 of this special register are driven to the G0 through G9 output pins.

The special registers described on this page are write-only registers, and reading any of them produces an undefined value.

INTERRUPTS

Register	Description
ISR	**Interrupt Status Register:** Contains the chip-status information. The ISR bits with the _sig (signal) postfix are included for simple host-LANai communication. A signal bit can be set only by the LANai processor and reset only from the EBUS, or vice versa. The ISR bits with the _rdy (ready) postfix are maintained by the packet interface, and cannot be modified by the programmer. However, when such a bit becomes 1, the packet interface may reset it only as a result of a bit-specific request to the packet interface. The ISR bits with the _int (interrupt) postfix are set by the packet interface or the EBUS interface when their corresponding events occur. These bits can be reset directly — by writing a 1 into them, or indirectly — in a bit-specific way.
IMR	**Interrupt Mask Register:** When a bit of ISR is equal to 1 and the corresponding bit of IMR is equal to 1, an interrupt request is asserted for the on-chip processor.
EIMR	**External-Interrupt Mask Register:** When a bit of ISR is equal to 1 and the corresponding bit of EIMR is equal to 1, the INT output pin is asserted.

Accessing some special registers has a side effect on the ISR. There is up to three-assembly-instructions delay between the time when a special register is accessed and the time of the change of the corresponding ISR bit(s). In case of tight polling loops, this delay can result in a race condition, whereby the program could, for example, misinterpret a not-yet-cleared ISR _int bit for an indication of a new event. The following tables specify which special-register access affects which ISR bits, and the maximum possible delay (in assembly instructions):

Writing	Clears	May Clear
DMA_CTR	dma_int (1)	.
IT	time_int (1) wdog_int (1)	.
RML	byte_rdy (2) half_rdy (2) word_rdy (2) buff_int (1) orun2_int (1) orun1_int (1) recv_int (1)	.
SML, SMLT	send_rdy (2) send_int (1)	.
SB, SH, SW, ST	.	send_rdy (3)

Reading	Clears	May Clear	May Set
RB, RH, RW	orun2_int (1) orun1_int (1) tail_int (1)	byte_rdy (3) half_rdy (3) word_rdy (3)	orun2_int (3) orun1_int (3) tail_int (3)

The ISR, IMR, and EIMR consist of the following bits (bit 31 is the most significant):

Bit	Name	Description
31	debug_bit	This bit is always equal to 1, and can be used for single-stepping the code that runs in user context (page ??).
30	host_sig	This bit is set when the LANai processor writes a 1 into it, and reset when a 1 is written into it from the EBUS.
29-24	0	Reserved.
23-16	lan7_sig - lan0_sig	Each of these 8 bits is set when a 1 is written into it from the EBUS, and reset when the LANai processor writes a 1 into it.
15	word_rdy	This bit is maintained by the packet interface and is equal to 1 if: (1) there are at least four flits of the same packet on the incoming channel, or (2) the first, the second, or the third available flit on the incoming channel is a tail. In either case, the word_rdy bit indicates that an RW operation can be issued. Reading RW while the word_rdy bit is equal to 0 may corrupt the incoming packet. During a receive-DMA operation (from the time when the RML is written, until the time when recv_int or buff_int becomes one), this bit is equal to 0 regardless of the state of the incoming channel. Note that word_rdy=1 implies that half_rdy=1 and that byte_rdy=1. However, because of (2), word_rdy=1 does not imply that, for example, two RH operations can be issued. This bit cannot be modified by the programmer.
14	half_rdy	This bit is maintained by the packet interface and is equal to 1 if: (1) there are at least two flits of the same packet on the incoming channel, or (2) the next available flit on the incoming channel is a tail. In either case, the word_rdy bit indicates that an RH operation can be issued. Reading RH while the half_rdy bit is equal to 0 may corrupt the incoming packet. During a receive-DMA operation (from the time when the RML is written, until the time when recv_int or buff_int becomes one), this bit is equal to 0 regardless of the state of the incoming channel. Note that half_rdy=1 implies that byte_rdy=1. However, because of (2), half_rdy=1 does not imply that two RB operations can be issued. This bit cannot be modified by the programmer.
13	send_rdy	This bit is maintained by the packet interface, and it denotes that the outgoing channel is not blocked, so that an SB, SH, SW, or an ST operation can be issued. Writing SB, SH, SW, or ST while this bit is equal to 0 may corrupt the outgoing packet. During a send-DMA operation (from the time when the SML(T) is written until the time when send_int becomes one), send_rdy is equal to 0 regardless of the state of the outgoing channel. This bit cannot be modified by the programmer.
12	0	Reserved.

Bit	Name	Description
11	nres_int	This bit is set by the Myrinet-SAN interface whenever the LANai chip fails to consume an incoming message from the Myrinet network for the duration of the period specified by the TIMEOUT special register. If the NRES_ENABLE bit of the MYRINET special register is 1, the LANai chip is also reset. By examining the nres_int bit, one can distinguish between the reset-pin-induced and NRES-induced reset. This bit is cleared by the programmer, only directly — by writing a 1 into it.
10	wake_int	This bit is set when the WAKE input pin is asserted. This bit is cleared by the programmer, only directly — by writing a 1 into it.
9 - 8	orun2_int orun1_int	These bits are set by the packet interface when an overrun condition is detected, i.e., when a receive operation reads past the tail flit of a packet. The two bits taken together represent the number of flits read past the tail flit (0 through 3). For example, if when using receive DMA (RMP and RML), the tail is received in the most-significant byte of a 4-byte word, both orun2_int and orun1_int will be set, indicating that the values of the 3 least-significant flits are undefined. These bits are cleared by the programmer, either directly — by writing a 1 into them, or indirectly — when any receive operation is initiated.
7	tail_int	This bit is set by the packet interface when an RB, an RH, or an RW operation consumes a tail flit. This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when another RB, RH, or RW operation is issued.
6	wdog_int	This bit is set by the interrupt timer whenever it makes a transition from 0x80000000 to 0x7FFFFFFF. This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when IT is written.
5	time_int	This bit is set by the interrupt timer whenever it makes a transition from 0x00000000 to 0xFFFFFFFF. This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when IT is written.
4	dma_int	This bit is set by the EBUS-LBUS DMA engine when the DMA_CTR reaches 0 to signal the completion of a DMA transfer. This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when the DMA_CTR is written. After an EBUS-LBUS DMA is initiated, one must not initiate another such DMA until the dma_int bit becomes 1.
3	send_int	This bit is set by the packet interface to signal the completion of a send DMA, i.e., when the contents of the send memory buffer and the tail CRC have been appended to the outgoing packet. This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when SML(T) is written. After a send DMA is initiated, one must not initiate another send DMA until the send_int bit becomes 1.
2	buff_int	This bit is set by the packet interface when the receive-DMA buffer has been exhausted (the last word written is at the location pointed to by RML, and RMP=RML+4). This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when RML is written. After a receive DMA is initiated, one must not initiate another receive DMA until the recv_int bit, the buff_int bit, or both, become 1.
1	recv_int	This bit is set by the packet interface to signal the completion of a receive DMA, i.e., when the entire incoming packet has been transferred into the receive memory buffer. This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when RML is written. After a receive DMA is initiated, one must not initiate another receive DMA until the recv_int bit, the buff_int bit, or both, become 1.
0	byte_rdy	This bit is maintained by the packet interface and is equal to 1 if there is at least one flit on the incoming channel, indicating that an RB operation can be issued. Reading RB while the byte_rdy bit is equal to 0 may corrupt the incoming packet. During a receive-DMA operation (from the time when the RML is written, until the time when recv_int or buff_int becomes one), this bit is equal to 0 regardless of the state of the incoming channel. This bit cannot be modified by the programmer.

SPECIAL-REGISTER SUMMARY

Register	Read		Write		Description	Offset	Page
	EBUS	LANai	EBUS	LANai			
CKS			+	+	Internet Checksum	0x38	5
CLOCK			+		Clock Configuration	0xFC	7
DEBUG			+	+	Hardware Debugging	0x90	7
DMA_CTR	+	+	+	+	Initiate EBUS DMA	0x44	5
DMA_DIR			+	+	Direction of EBUS DMA	0x80	5
DMA_STS			+	+	EBUS-DMA Configuration	0x84	5
EAR	+	+	+	+	EBUS-DMA Host Address	0x3C	5
EIMR	+	+	+	+	External-Interrupt Mask	0x2C	8
IMR		+		+	LANai-Interrupt Mask	-	8
ISR	+	+	+	+	Interrupt Status	0x28	8
IT	+	+	+	+	Interrupt Timer	0x30	6
LAR	+	+	+	+	EBUS-DMA LBUS address	0x40	5
LED			+	+	LED Output Pin	0x94	7
MP			+	+	Memory Protection	0x9C	6
MYRINET			+	+	Myrinet-Link Configuration	0x8C	7
RB	+	+			Receive Byte	0x60	4
RH	+	+			Receive Half-Word	0x64	4
RML	+	+	+	+	Initiate Receive DMA	0x4C	4
RMP	+	+	+	+	Receive-DMA Buffer	0x48	4
RTC	+	+	+	+	Real-Time Clock	0x34	6
RW	+	+			Receive Word	0x68	4
SA			+	+	Send-DMA Alignment	0x6C	3
SB			+	+	Send Byte	0x70	3
SH			+	+	Send Half-Word	0x74	3
SML	+	+	+	+	Initiate Send DMA	0x54	3
SMLT	+	+	+	+	Initiate Send DMA with Tail	0x58	3
SMP	+	+	+	+	Send-DMA Buffer	0x50	3
ST			+	+	Send Tail	0x7C	3
SW			+	+	Send Word	0x78	3
TIMEOUT			+	+	NRES-Timeout Selection	0x88	7
WINDOW			+	+	Sampling-Window Selection	0x98	7

All special registers except for the IMR are memory-mapped. The IMR is an internal register of the LANai on-chip processor and is not accessible from the EBUS. The memory-mapped special registers can be accessed both by the LANai on-chip processor and from the EBUS (except for the special register CLOCK, that can be accessed only from the EBUS).

To access a memory-mapped special register from the LANai processor one should use the address of 0xFFFFFF00 plus the offset of that special register. The base address for EBUS access of memory-mapped special registers is application-specific; consult system documentation for details.

When accessing the special, memory-mapped registers, the regular memory arbitration mechanism described on page 2 applies. The mutual exclusion at any higher level is the responsibility of the programmer.

INITIALIZATION

During the power-on reset, the chip-version-specific value listed below must be written to the CLOCK special register from the EBUS, to initialize the on-chip clock generation.

After chip reset, the on-chip processor begins executing code in the system context, starting from the address 0.

The state of the nres_int bit in the ISR upon reset indicates whether the reset has been a regular, reset-pin initialization (0), or an NRES-induced reset (1).

All the remaining bits of ISR are equal to 0, except the debug_bit and the send_rdy bit, which are equal to 1.

The MP register is initialized to the no-memory-protection state.

The IMR and the EIMR special registers are undefined and should be initialized by the programmer.

All other special registers are initialized to 0 upon reset.

After the chip is out of reset and prior to any Myrinet access, the system-specific value listed below must be written to the WINDOW register to configure the Myrinet-SAN interface, and a minimum of 10 milliseconds must be allowed for the SAN link to reconfigure.

CHIP-VERSION-SPECIFIC INITIALIZATION

Version	CLOCK	WINDOW
LANai4.0	0x11371137	0
LANai4.1	0x50E450E4	3
LANai4.2	0x90449044	3
LANai4.3	0x90449044	3

SIMPLE MESSAGE SENDING AND RECEIVING EXAMPLES

In all the examples, data is represented by the following symbols:

```
'm'   -  message byte
'.'   -  don't-care byte
'c8'  -  CRC-8 byte
'v8'  -  a verified CRC-8 byte (zero if CRC-8 caught no errors)
```

Example #1:

```
        CRC-8   off
                                                 sender memory
                                               msb             lsb
sender ()                                      +---+---+---+---+
{                                     SMP ->   | m | m | m | m |
  SMP  = 0x1000;                    (start)    +---+---+---+---+
  SMLT = 0x1008;                               | m | m | m | m |
}                                              +---+---+---+---+
                                      SMLT ->  | m | m | m | m |
                                               +---+---+---+---+
                                      SMP ->   | . | . | . | . |
                                     (end)     +---+---+---+---+


                                               bytes in Myrinet
                                               +---+---+---+---+
                                               | m | m | m | m |
                                               +---+---+---+---+
                                               | m | m | m | m |
                                               +---+---+---+---+
                                               | m | m | m | m |
                                               +---+---+---+---+
                                               | 0 |
                                               +---+

                                                 receiver memory
                                               msb             lsb
receiver ()                                    +---+---+---+---+
{                                     RMP ->   | m | m | m | m |
  RMP = 0x2000;                     (start)    +---+---+---+---+
  RML = 0x3000;                                | m | m | m | m |
}                                              +---+---+---+---+
                                               | m | m | m | m |
                                               +---+---+---+---+
                                               | 0 | . | . | . |
                                               +---+---+---+---+
                                      RMP ->   | . | . | . | . |
                                     (end)     +---+---+---+---+
                                                 .   .   .   .
                                                 .   .   .   .
                                                 .   .   .   .
                                               +---+---+---+---+
                                      RML ->   | . | . | . | . |
                                               +---+---+---+---+
```

Example #2:

```
        CRC-8    on
```

```
sender ()
{
  SMP  = 0x1000;
  SMLT = 0x1008;
}
```

```
              sender memory
           msb              lsb
           +---+---+---+---+
  SMP  ->  | m | m | m | m |
(start)    +---+---+---+---+
           | m | m | m | m |
           +---+---+---+---+
  SMLT ->  | m | m | m | m |
           +---+---+---+---+
  SMP  ->  | . | . | . | . |
 (end)     +---+---+---+---+
```

```
           bytes in Myrinet
           +---+---+---+---+
           | m | m | m | m |
           +---+---+---+---+
           | m | m | m | m |
           +---+---+---+---+
           | m | m | m | m |
           +---+---+---+---+
           | c8|
           +---+
```

```
receiver ()
{
  RMP = 0x2000;
  RML = 0x3000;
}
```

```
             receiver memory
           msb              lsb
           +---+---+---+---+
  RMP  ->  | m | m | m | m |
(start)    +---+---+---+---+
           | m | m | m | m |
           +---+---+---+---+
           | m | m | m | m |
           +---+---+---+---+
           | v8| . | . | . |
           +---+---+---+---+
  RMP  ->  | . | . | . | . |
 (end)     +---+---+---+---+
             .   .   .   .
             .   .   .   .
             .   .   .   .
           +---+---+---+---+
  RML  ->  | . | . | . | . |
           +---+---+---+---+
Example #3:

        CRC-8    on
        SA       used
                                           sender memory
                                         msb            lsb
sender ()                                +---+---+---+---+
{                                SMP ->  | . | m | m | m |
  SMP  = 0x1000;               (start)   +---+---+---+---+
  SA   = 0x1;                            | m | m | m | m |
  SMLT = 0x1008;                         +---+---+---+---+
}                                SMLT -> | m | m | m | m |
                                         +---+---+---+---+
                                 SMP ->  | . | . | . | . |
                                 (end)   +---+---+---+---+


                                         bytes in Myrinet
                                             +---+---+---+
                                             | m | m | m |
                                         +---+---+---+---+
                                         | m | m | m | m |
                                         +---+---+---+---+
                                         | m | m | m | m |
                                         +---+---+---+---+
                                         | c8|
                                         +---+


                                          receiver memory
                                         (assuming the first
                                         three bytes have been
                                         stripped by switches)
                                         msb            lsb
                                         +---+---+---+---+
receiver ()                      RMP ->  | m | m | m | m |
{                               (start)  +---+---+---+---+
  RMP = 0x2000;                          | m | m | m | m |
  RML = 0x3000;                          +---+---+---+---+
}                                        | v8| . | . | . |
                                         +---+---+---+---+
                                 RMP ->  | . | . | . | . |
                                 (end)   +---+---+---+---+
                                           .   .   .   .
                                           .   .   .   .
                                           .   .   .   .
                                         +---+---+---+---+
                                 RML ->  | . | . | . | . |
                                         +---+---+---+---+
```

ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS

Symbol	Rating	Value	Unit
V_{dd}	Power Supply Voltage	-0.5 to +4.6	V
V_{in}, V_{out}	Terminal Voltage (except Vdd)	-0.5 to V_{dd}+0.5	V
I_{out}	Output Current	100	mA
P_D	Power Dissipation	3	W
T_{bias}	Temperature Under Bias	-55 to 125	°C
T_{stg}	Storage Temperature	-55 to 125	°C
T_A	Operating Temperature	0 to 70	°C

RECOMMENDED OPERATING CONDITIONS

Symbol	Parameter	Min	Typ	Max	Unit
V_{dd}	Power Supply Voltage	3.0	3.3	3.6	V
V_{IH}	Input High Voltage	2.2	-	Vdd+0.3	V
V_{IL}	Input Low Voltage	-0.3	-	0.8	V

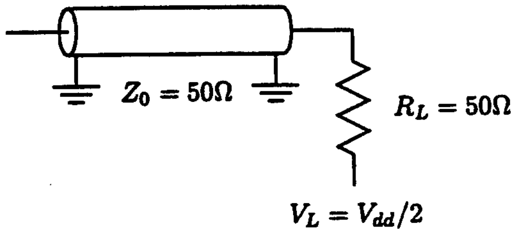
DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Unit
I_{LI}	Input Leakage Current	-	±1.0	μA
I_{LO}	Output Leakage Current	-	±1.0	μA
$V_{OL}(IOL = 5mA)$	Output Low Voltage	-	0.4	V
$V_{OH}(IOH = -5mA)$	Output High Voltage	2.4	-	V

CAPACITANCE ($T_A = +25°C, f = 1.0MHz, dV = 3V$)

Symbol	Parameter	Max	Unit
C_I	Input Capacitance	5	pF
$C_{I/O}$	I/O Capacitance	8	pF

AC TEST LOADS



OUTPUT DRIVERS

The output drivers and the tri-state drivers are of three different strengths. For purely capacitive loads, the following are the typical delay values at $V_{dd} = 3.3V, T_a = 25^oC$:

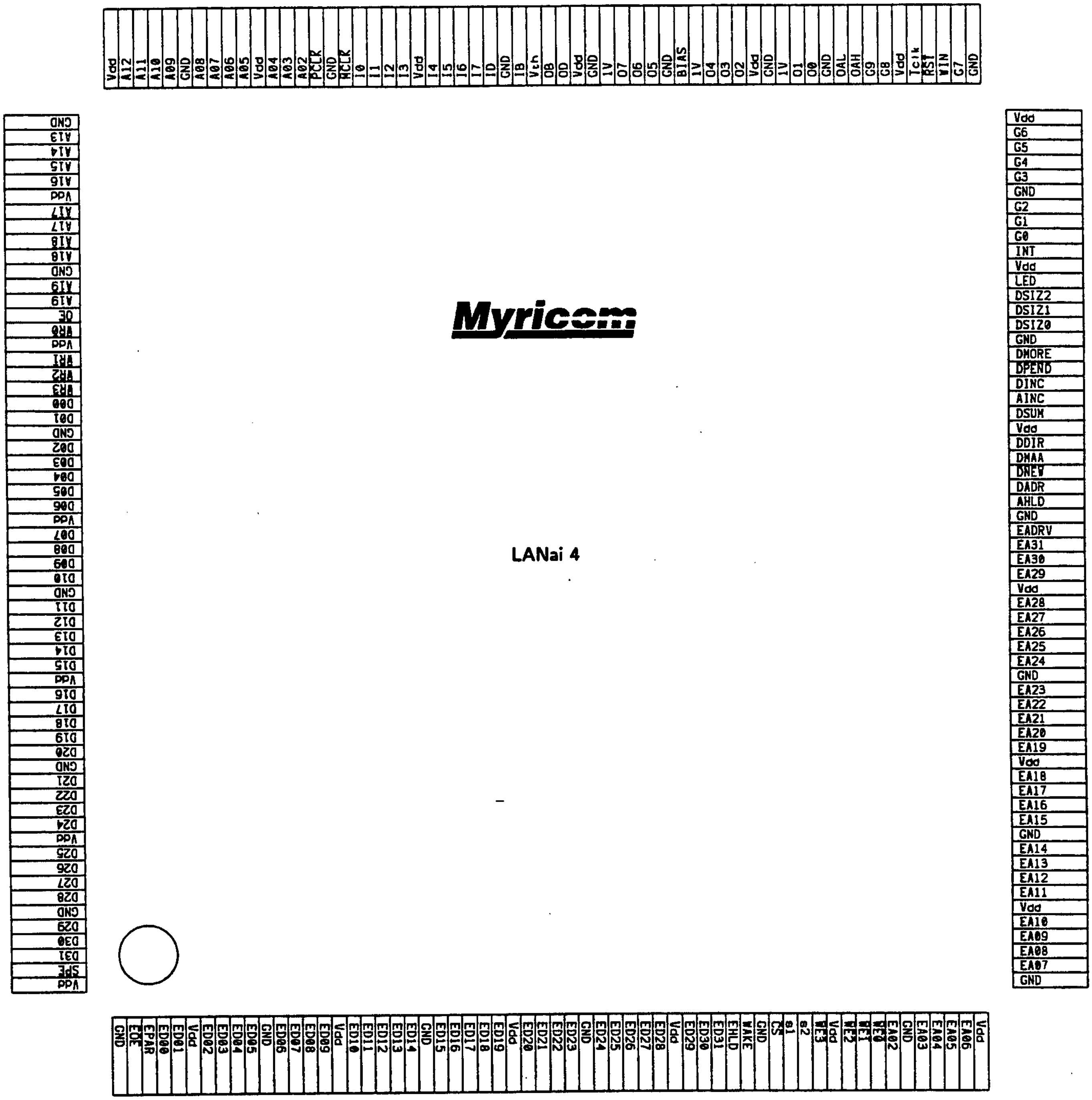
Output	Delay
A02 - A19, A18, A19, A20, OE	$out_L = 0.2ns + C_{\rm load}[pF] * 0.024ns/pF$
EA02 - EA31, ED00 - ED31	$out_E = 0.2ns + C_{\rm load}[pF] * 0.030ns/pF$
otherwise	$out = 0.2ns + C_{\rm load}[pF] * 0.042ns/pF$

For power-supply voltage of $3.3V \pm 10\%$, ambient temperature from 0^o to 70^oC, and the acceptable manufacturing-process variation, output-driver delays have $\pm 50\%$ variation.

The full hspice model is available upon request.

Note: The LANai4.1 version of the chip has weaker output drivers. Data available upon request.

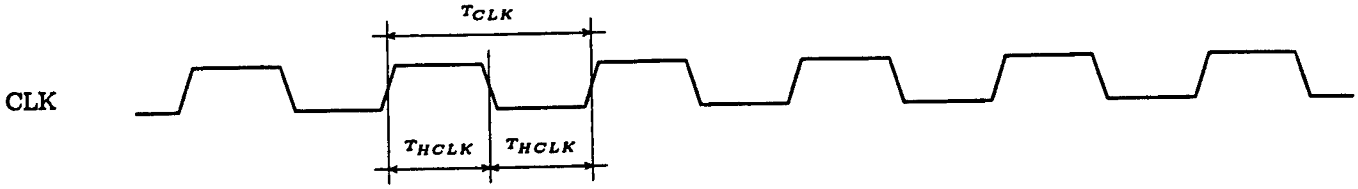
PINOUT



240-pin QFP (top view)

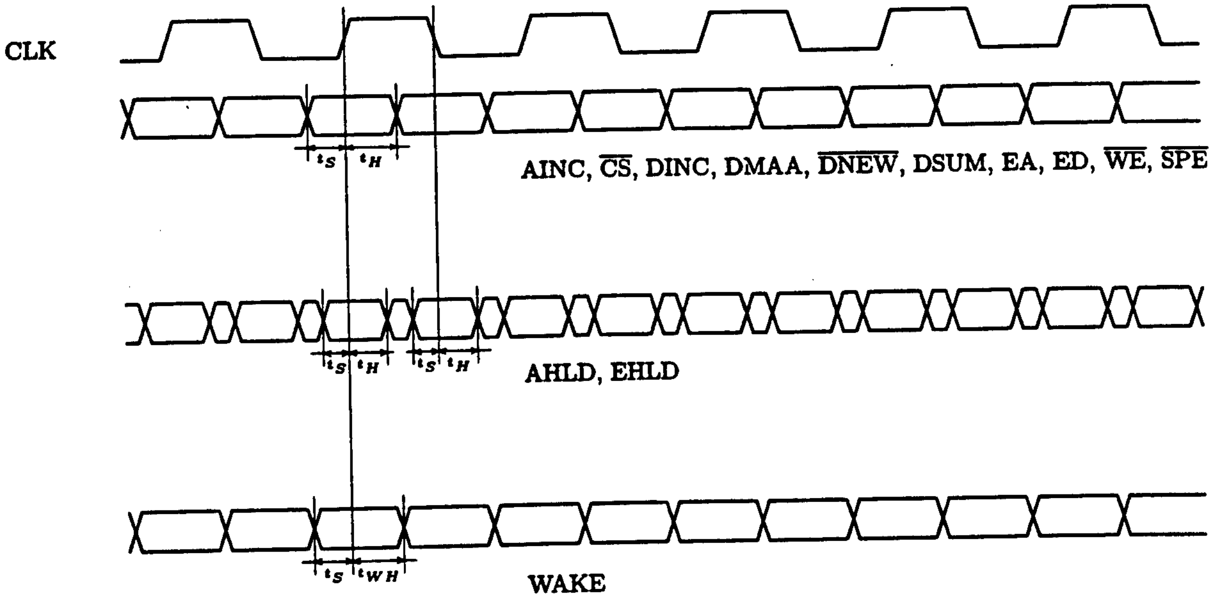
CLOCKING

Pin	I/O	Description
CLK	I	Clock: The main clock input.



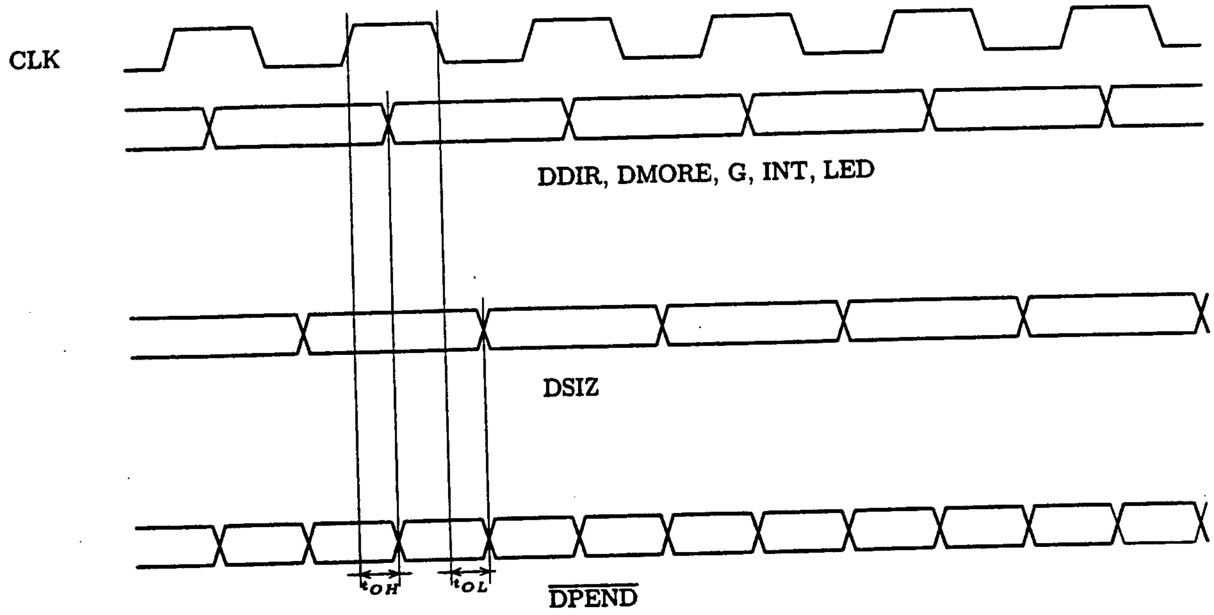
Symbol	Parameter	Min	Max
T_{CLK}	Clock period	25ns	20μs
T_{HCLK}	Clock half-period	12ns	10μs

SYNCHRONOUS INPUTS


CLK
t_S
t_H
AINC, $\overline{CS}$, DINC, DMAA, $\overline{DNEW}$, DSUM, EA, ED, $\overline{WE}$, $\overline{SPE}$
t_S
t_H
t_S
t_H
AHLD, EHLD
t_S
t_{WH}
WAKE

Symbol	Parameter	Min	Max
t_S	Synchronous-input setup time	1ns	
t_H	Synchronous-input hold time	3ns	
t_{WH}	WAKE hold time	5ns	

SYNCHRONOUS OUTPUTS

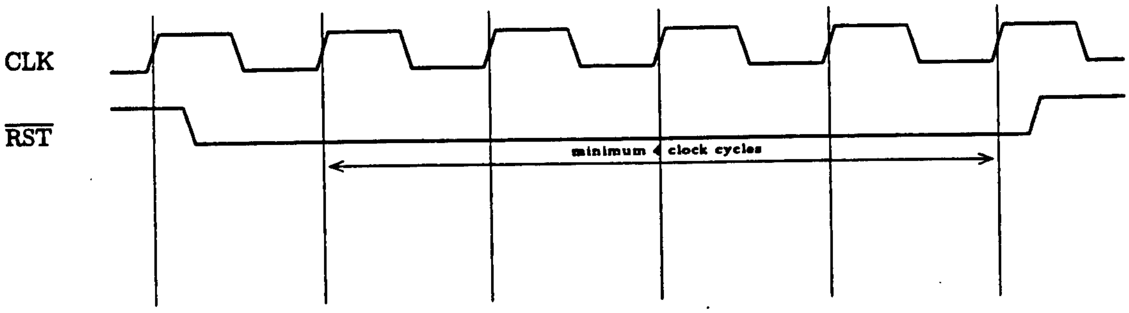


Symbol	Parameter	Min	Max
t_{OH}	Synchronous-output delay	0.5ns	8ns
t_{OL}	Synchronous-output delay	0.5ns	8ns

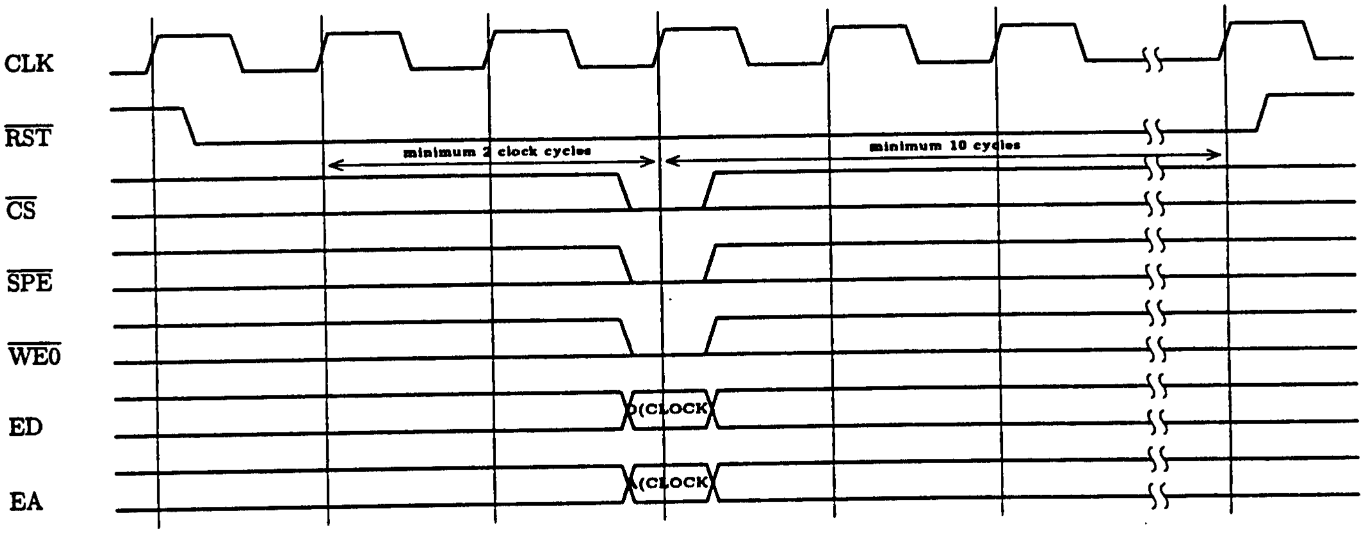
RESET SEQUENCE

Pin	I/O	Description
$\overline{\text{RST}}$	I	**Reset:** The main reset input. During the power-on reset, the special register CLOCK, which controls the on-chip clock-generation, must be initialized. Note that, unlike with other special registers, $\overline{\text{RST}}$ must be asserted while the CLOCK is written. This register need not be initialized during a non-power-on reset.

Non-Power-On Reset Sequence



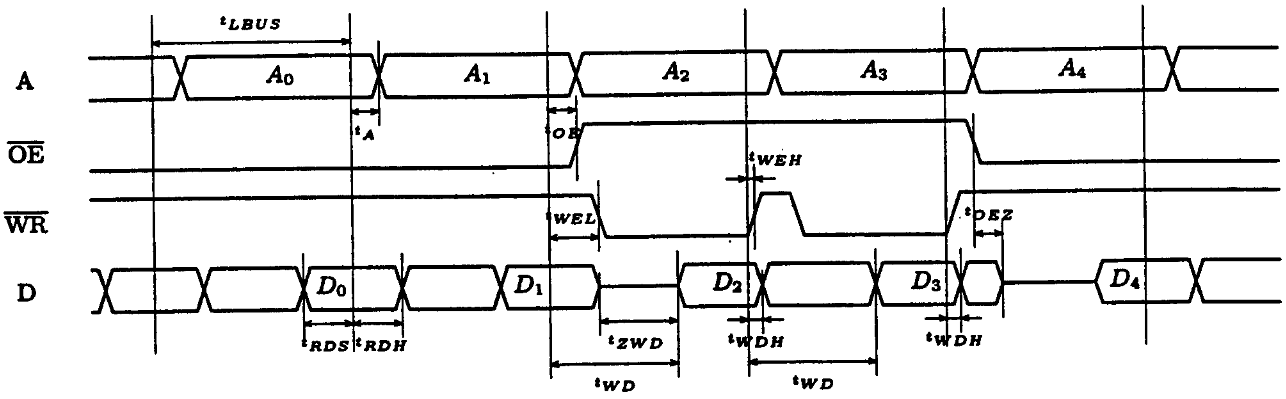
Power-On Reset Sequence



Note: For LANai4.1 chips, the CLOCK register must be initialized with the value 0x50E450E4

LBUS INTERFACE

Pin	I/O	Description
D00 - D31	I/O	**LBUS Data:** 32-bit bi-directional data bus (bit 31 is the most significant).
A02 - A19	O	**LBUS Address:** The LBUS address pins. The internals of the LANai 4 chip support 32-bit addresses, but pin count limits the LBUS address space to 1 megabyte.
$\overline{\text{A17}}$ - $\overline{\text{A19}}$	O	**LBUS Address Complement:** These three signals are provided for address decoding when using multiple banks of SRAM chips.
$\overline{\text{WR0}}$ - $\overline{\text{WR3}}$	O	**LBUS Write Enable:** 8 write enable signals, one for each byte in the 64-bit word. The LANai is a big-endian machine, and $\overline{\text{WE0}}$ corresponds to the smallest byte address, i.e., to the most-significant byte.
$\overline{\text{OE}}$	O	**LBUS Output Enable:** This signal controls the direction of the LBUS data bus.



Symbol	Parameter	Min	Max
t_{LBUS}	LBUS memory cycle	$T_{HCLK} - 0.5$ns	
t_A	Address delay	out_L	$out_L + 0.5$ns
t_{OE}	Output-enable delay	out_L	$out_L + 0.5$ns
t_{WEL}	Write-enable assertion delay	$out + 2$ns	$out + 3$ns
t_{WEH}	Write-enable deassertion delay	out	$out + 0.5$ns
t_{RDS}	Read-data setup time	3ns	
t_{RDH}	Read-data hold time	1ns	
t_{WD}	Write-data delay (LANai limited)		$out + 4$ns
t_{ZWD}	Write-data delay (SRAM limited)		out
t_{WDH}	Write-data hold time	out	
t_{OEZ}	Data-drivers turn-off time	0ns	0.5ns

EBUS INTERFACE

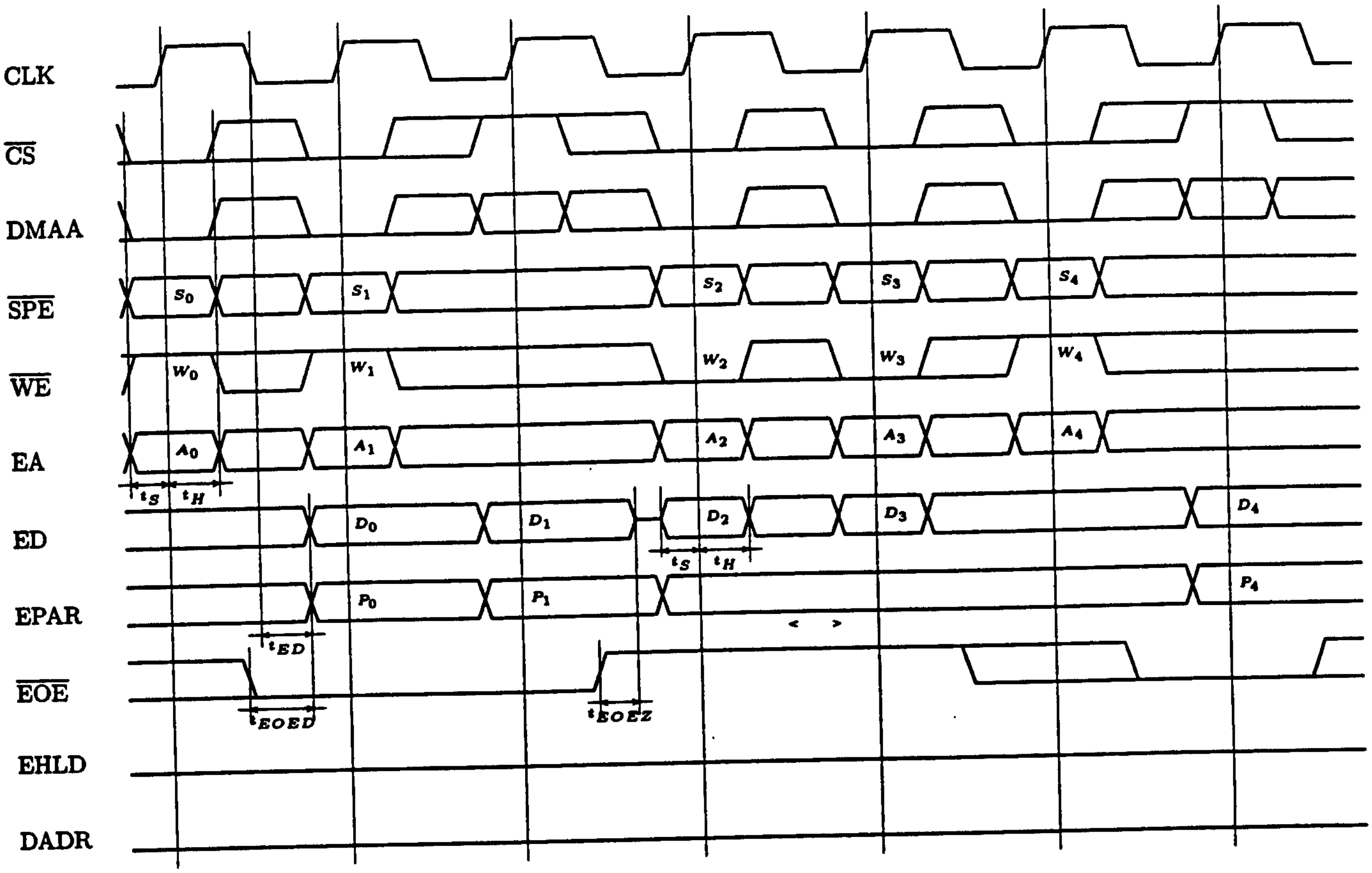
The LANai 4 EBUS is a 32-bit wide, synchronous interface.

Core EBUS functionality provides for reading and writing the LANai LBUS memory, and the LANai special registers. The rest of the EBUS interface contains the circuitry that can assist the external hardware in implementing the EBUS DMA: LBUS address register, EBUS address register, access counter, and mechanisms for controlling these registers in sync with the data transfer.

EBUS SINGLE-ACCESS MODE

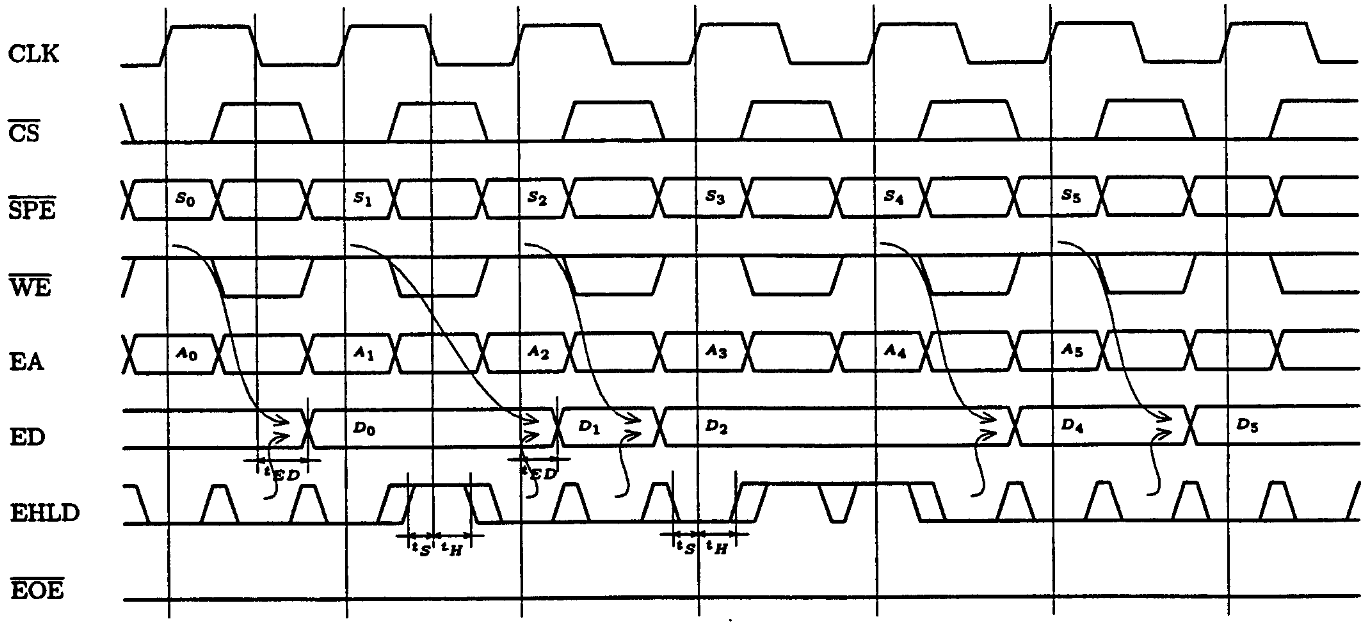
Pin	I/O	Description
$\overline{\text{CS}}$	synch. I	**EBUS Chip Select:** When this pin is asserted on a rising CLK edge, the first of the two memory cycles during the immediately following clock cycle will be used for EBUS access (page 2).
DMAA	synch. I	**DMA Address:** If this pin is not asserted, the current EBUS access is a single LBUS memory access or a single special-register access, with the LBUS address specified on the EA pins. If this pin is asserted, the current EBUS access is part of a DMA, and special register LAR is used to provide the LBUS address.
$\overline{\text{SPE}}$	synch. I	**EBUS Special Register Access:** In single-access EBUS mode, if this pin is asserted a special register is accessed, and if it is not asserted the LBUS memory is accessed.
$\overline{\text{EOE}}$	I	**Output Enable For ED Pins:** When this signal is asserted, the ED bus is the output.
ED00 - ED31	synch. I/O	**EBUS Data:** Bi-directional EBUS data pins (bit 31 is the most significant).
EPAR	O	**EBUS Parity:** This output is computed as the parity of the 32 ED pins.
EHLD	synch. I	**ED Hold:** During EBUS read access, the EHLD input determines the timing of the ED bus. In the typical operating regime, EHLD is never asserted and the data changes when CLK is low, to be sampled by the external hardware on the rising CLK edge. If the external hardware cannot latch the data as described above, it can hold the read data on the ED bus as long as necessary by keeping the EHLD input asserted: The EHLD is sampled on each CLK edge to determine if ED bus is allowed to change during the upcoming half of clock cycle.
EA02 - EA31	synch. I/O	**EBUS Address:** Bi-directional EBUS address pins (bit 31 is the most significant). In single-access mode (DMAA not asserted), if $\overline{\text{SPE}}$ is not asserted, pins EA02-EA19 specify the address of the LBUS memory, or, if $\overline{\text{SPE}}$ is asserted, pins EA02-EA08 select a LANai special register. In DMA mode (DMAA asserted), these pins are used to output the address specified by the special register EAR (EBUS address register).
$\overline{\text{WE0}}$ - $\overline{\text{WE3}}$	synch. I	**EBUS Write Enable:** Write enable signals, one for each byte in the 32-bit word. The LANai is a big-endian machine, and $\overline{\text{WE0}}$ corresponds to the smallest byte address, i.e., to the most-significant byte. When accessing LANai special registers, $\overline{\text{WE0}}$ determines if the access is a read or a write.

Single-Access EBUS Timing



Symbol	Parameter	Min	Max
t_{ED}	Read-data access time (LANai limited)	out_E	out_E + 3ns
t_{EOED}	Read-data access time (EOE limited)	out_E	out_E + 2ns
t_{EOEZ}	Data-bus release time	0ns	2ns

EHLD Timing

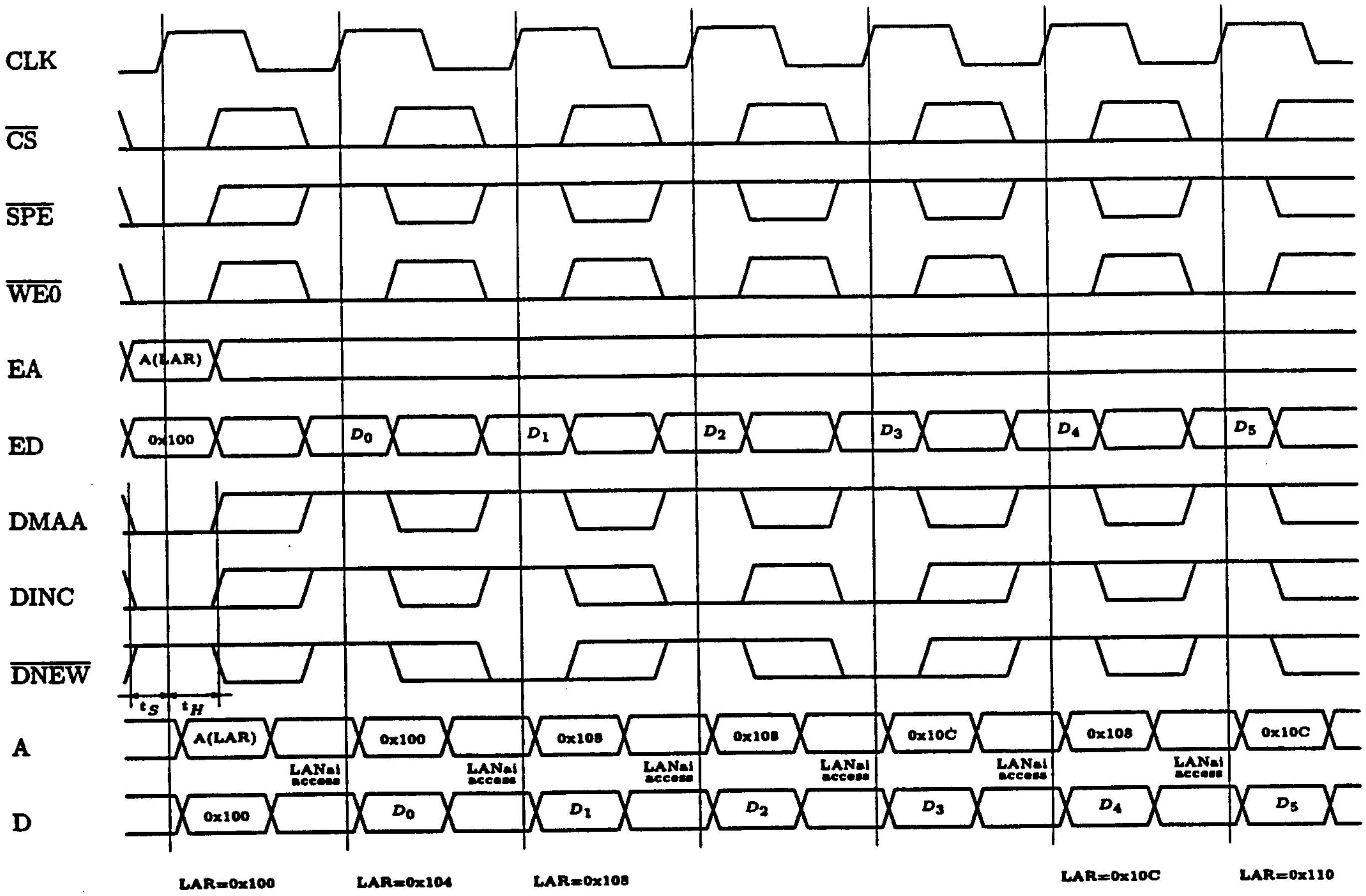


CLK
$\overline{\text{CS}}$
$\overline{\text{SPE}}$
$\overline{\text{WE}}$
EA
ED
EHLD
$\overline{\text{EOE}}$
S_0
S_1
S_2
S_3
S_4
S_5
A_0
A_1
A_2
A_3
A_4
A_5
D_0
D_1
D_2
D_4
D_5
t_{ED}
t_{ED}
t_S
t_H
t_S
t_H

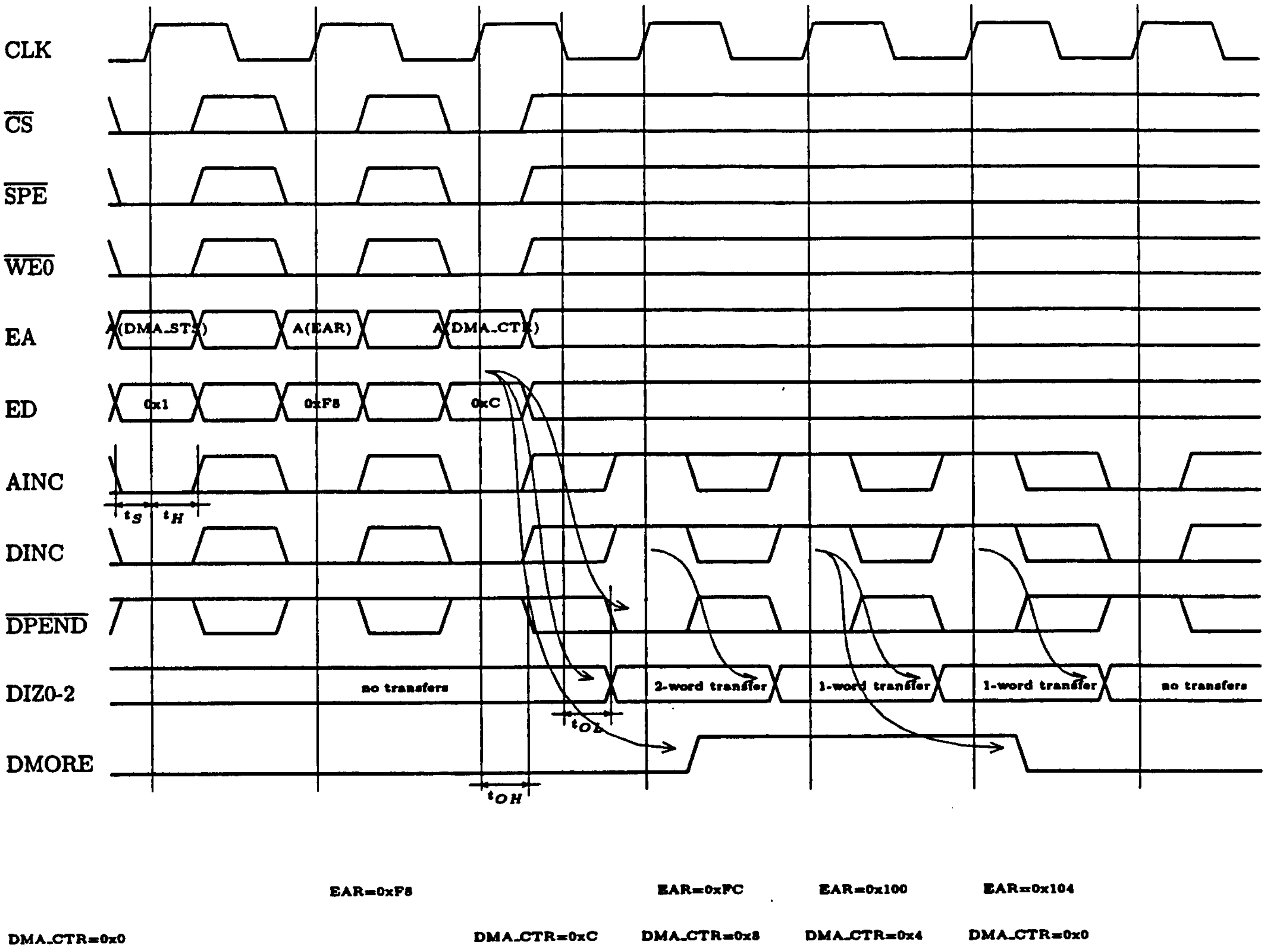
EBUS DMA MODE

<table>
<tr><th>Pin</th><th>I/O</th><th>Description</th></tr>
<tr><td>AINC</td><td>synch.
I</td><td><b>Advance DMA Register EAR:</b> When this input is asserted, special register EAR is incremented by 4.</td></tr>
<tr><td>DINC</td><td>synch.
I</td><td><b>Advance DMA Registers LAR and DMA_CTR:</b> When this input is asserted, special register LAR is incremented by 4, and special register DMA_CTR is decremented by 4.</td></tr>
<tr><td>$\overline{\text{DNEW}}$</td><td>synch.
I</td><td><b>Pre-Increment LAR:</b> During EBUS DMA, the DINC input is used to increment the LBUS address register (LAR). The $\overline{\text{DNEW}}$ input selects if the LBUS address for the current DMA memory access should be the value of LAR before incrementing (if $\overline{\text{DNEW}}$ is not asserted) or after incrementing (if $\overline{\text{DNEW}}$ is asserted).</td></tr>
<tr><td>DSUM</td><td>synch.
I</td><td><b>DMA Checksum:</b> When this input is asserted, the data of the current EBUS access is included in the current Internet-checksum computation (special register CKS).</td></tr>
<tr><td>$\overline{\text{DPEND}}$</td><td>synch.
O</td><td><b>DMA Pending:</b> This signal is asserted when the special register DMA_CTR has a non-zero value.</td></tr>
<tr><td>DSIZ0
-
DSIZ2</td><td>synch.
O</td><td><b>DMA Size:</b> Some standard I/O buses (such as SBus) have burst-transfer modes that require that each data block be of size 2^N bytes, starting on 2^N-aligned address.
The special register DMA_STS (page 5) specifies the allowed transfer modes for the I/O bus that the LANai connects to.
The DSIZ signals encode (using the SBus convention) the size of the largest aligned block that can be transferred next, given the current values of EAR, DMA_CTR, and DMA_STS.
<table>
<tr><th>DSIZ2</th><th>DSIZ1</th><th>DSIZ0</th><th>Pending Transfer</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1 word</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>N/A</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>N/A</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>no transfers</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>4 words</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>8 words</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>16 words</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>2 words</td></tr>
</table></td></tr>
<tr><td>DMORE</td><td>synch.
O</td><td><b>DMA Pending:</b> This signal is asserted to indicate that there is at least one more burst-transfer following the one currently specified by the DSIZ pins.</td></tr>
<tr><td>EADRV</td><td>I</td><td><b>Output Enable For EA Pins:</b> When this signal is asserted, the EA bus is the output.</td></tr>
<tr><td>AHLD</td><td>synch.
I</td><td><b>EA Hold:</b> During EBUS DMA, the AHLD input determines the timing of the EA bus.
In the typical operating regime, AHLD is never asserted and the data is driven when CLK is low, sampled by the external hardware on the rising CLK edge.
If the external hardware cannot latch the data as described above, it can hold the read data on the EA bus as long as necessary by keeping the AHLD input asserted: The AHLD is sampled on each CLK edge to determine if EA bus is allowed to change during the upcoming half of clock cycle.</td></tr>
<tr><td>DADR</td><td>I</td><td><b>Drive EBUS Address Onto the ED bus:</b> In some systems it may be more convenient to obtain the value of the EAR on the ED, rather than on the EA bus. When DADR is asserted, the data presented on the ED bus is the current value of EAR, rather than the data of the current EBUS access.</td></tr>
</table>

DMAA, DINC, $\overline{\text{DNEW}}$ Timing

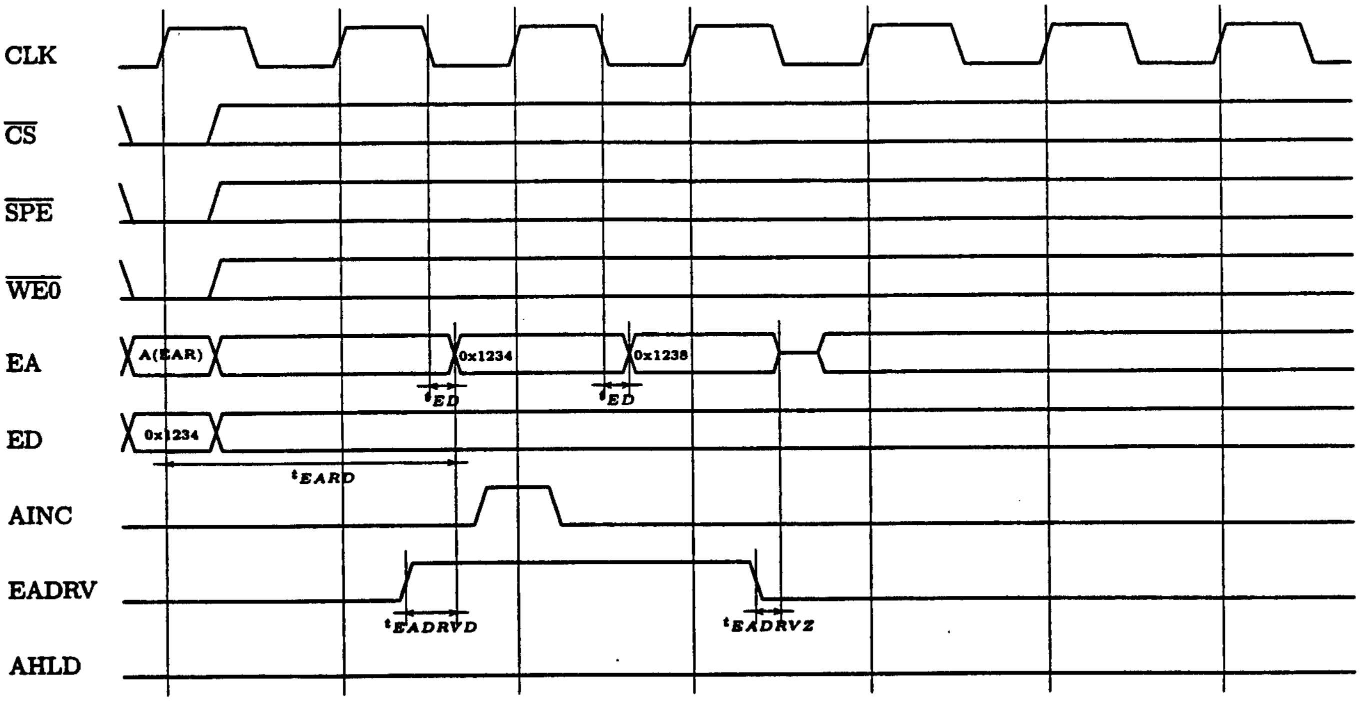


$\overline{\text{DPEND}}$, DSIZ0-DSIZ2, DMORE, AINC Timing



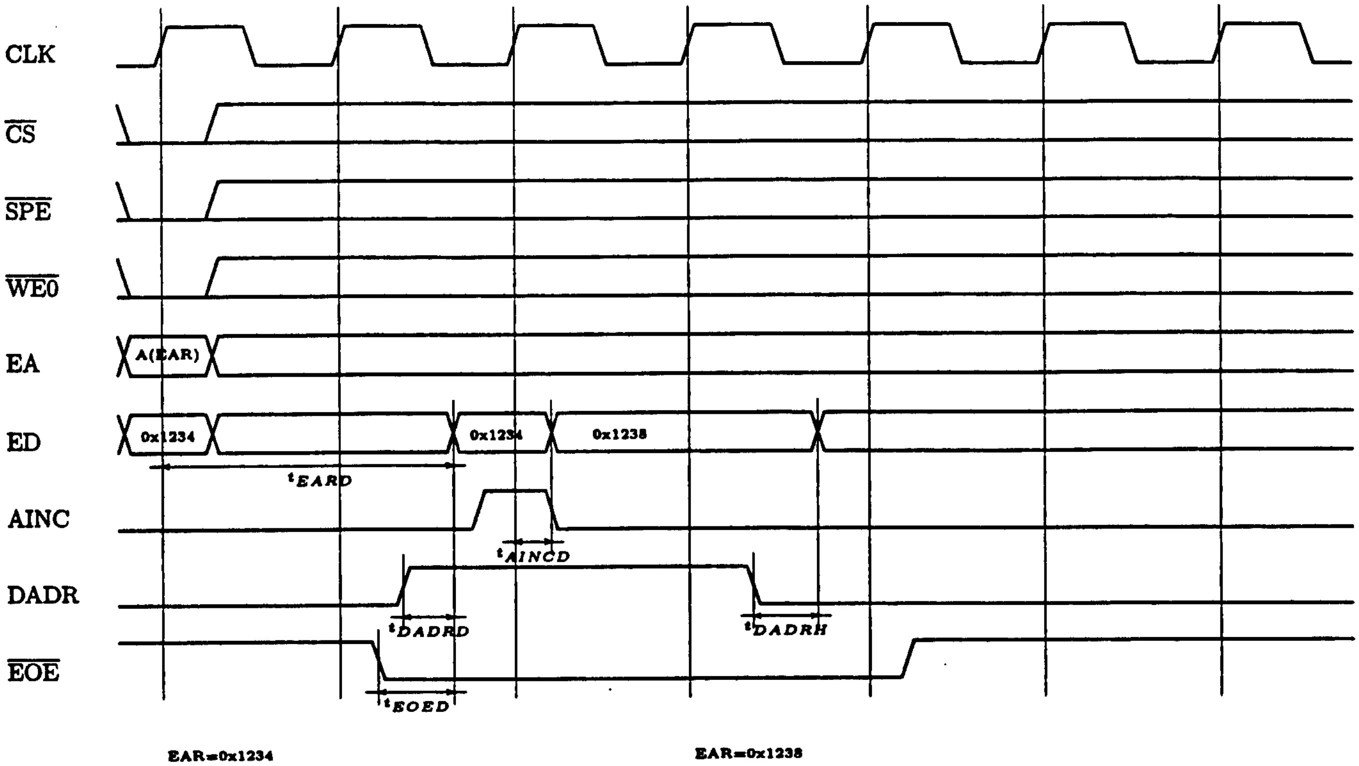
Note: The example value of DMA_STS enables only 2-word bursts

EADRV Timing



Symbol	Parameter	Min	Max
t_{EARD}	EAR access time		$out_E + 10$ns
t_{EADRVD}	EADRV access time		$out_E + 2$ns
t_{EADRVZ}	Address bus release time	0ns	2ns

DADR Timing



Symbol	Parameter	Min	Max
t_{DADRD}	DADR access time		out_E + 2ns
t_{DADRH}	DADR hold time	0ns	
t_{AINCD}	AINC access time		out_E + 5ns

Pin	I/O	Description
WIN	O	**Chip Window:** This signal should be connected to a test point that can be used for timing observation of several internal chip signals. The special register DEBUG selects which internal signal is to be output on the WIN pin.
LED	synch. O	**LED Output:** This general-purpose output is controlled by the least-significant bit of the special register LED.
G0 - G9	synch. O	**General-Purpose Output:** These general-purpose outputs are controlled by the bits 1 through 10 of the special register LED.
DDIR	synch. O	**EBUS DMA Direction:** This general-purpose output is the complement of the least-significant bit of the special register DMA_DIR (page 5). It is typically used to specify the direction of the upcoming EBUS DMA (asserted DDIR specifies the LBUS→EBUS direction).
INT	synch. O	**Interrupt Request:** This output is asserted if a bit in the special register ISR (Interrupt Status Register) and the corresponding bit in the special register EIMR (External Interrupt Mask Register) are both equal to 1.
WAKE	synch. I	**Wakeup:** When this input is asserted, the wake_int bit in the special register ISR (Interrupt Status Register) is set.

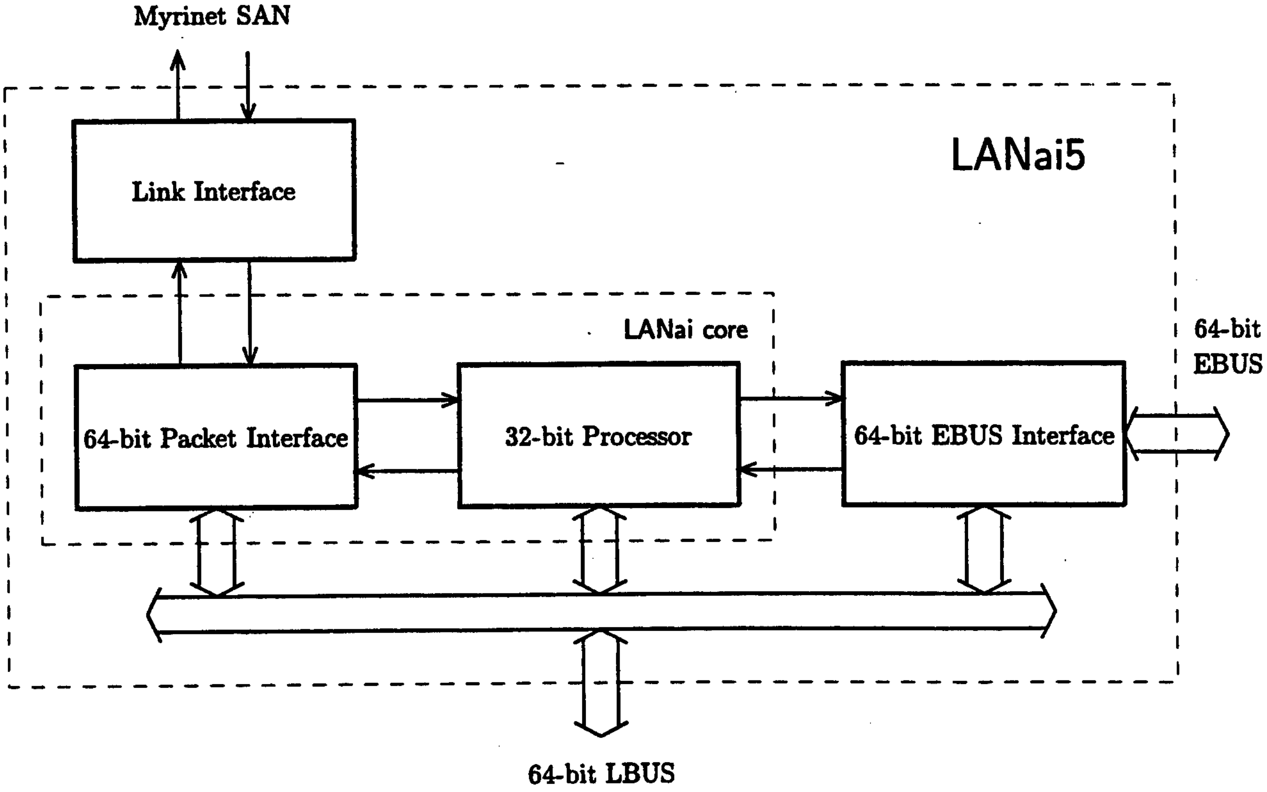
MYRINET SAN INTERFACE

	Pin	I/O	Description
Configuration	TCLK	I	**SAN Transmit Clock:** This clock input sets the data rate of the SAN output link. A byte is sent on every transition, so, for the nominal Myrinet SAN link rate of 160 megabytes per second, this is an 80 MHz clock. The allowed range for duty cycle is 45-55%. There is no restriction on the phase of TCLK with respect to any other clock (including TCLK on the other end of the SAN link). The frequency of TCLK must be within 1000ppm (0.1%) of 80MHz for compatibility with other Myrinet products.
	LVDD	power	**SAN Low-Voltage-Driver-Supply Voltage:** This pin should be connected to a 1.25V +/- 2% supply if it is to be compatible with other Myrinet SAN links. Power consumption varies with channel usage, 20mA minimum, 90mA maximum. If the SAN output drivers are shorted to GND, current draw can exceed 240mA.
	VTH	I	**SAN Input-Threshold Reference:** This reference input should be LVDD/2 ± 1%. Current draw is $1\mu A$ max.
	BIAS	I	**SAN Bias Reference:** For 3.3V operation, this pin should be fed 1.0mA. At that current level, the voltage on the pin will be approximately 1.4V. A 1.87KΩ resistor to 3.3V supply will achieve this.
	OAH	SAN O	**SAN Impedence Reference, High:** This pin should be connected to GND through a 50 ohm resistor.
	OAL	SAN O	**SAN Impedence Reference, Low:** This pin should be connected to LVDD through a 50-ohm resistor.
Input Channel	I0 - I7 ID	SAN I	**SAN Input:** The 8 data bits (I0-I7, I7 most significant) and the control bit (ID) are transition encoded. A transition on a data bit corresponds to the value of 1, no transition to the value of 0. A transition on the control bit corresponds to the data byte, no transition to the control symbol (see Myrinet SAN Link Specification). Each of these pins have a built-in 20KΩ pulldown resistor.
	OB	SAN O	**SAN Output Block:** This output is asserted to notify the connecting output SAN link channel that it must stop transmitting (see Myrinet SAN Link Specification).
Output Channel	O0 - O7 OD	SAN O	**SAN Output:** The 8 data bits (O0-O7, O7 most significant) and the control bit (OD) are transition encoded. A transition on a data bit corresponds to the value of 1, no transition to the value of 0. A transition on the control bit corresponds to the data byte, no transition to the control symbol (see Myrinet SAN Link Specification).
	IB	SAN I	**SAN Input Block:** When this input is asserted, the output channel stops transmitting (see Myrinet SAN Link Specification). This pin has a built-in 20KΩ pulldown resistor.

The LANai 5 is a programmable communication device that provides an interface to the Myrinet system-area network (SAN).

As illustrated below, a LANai 5 chip consists of the LANai core, with an instruction-interpreting processor and a packet interface, the Myrinet-SAN interface, and the EBUS interface.

The Local Bus (LBUS) is an interface to asynchronous static SRAMs. The External Bus (EBUS) is a synchronous, pipelined interface.



MEMORY INTERFACE

In the remainder of this specification, we shall refer to 8-bit data units as bytes, to 16-bit units as half-words, to 32-bit units as words, and to 64-bit units as double-words. Although the internals of the LANai 5 chip support 32-bit addresses, pin-count limitations restrict the LBUS to a maximum of 16M bytes.

Depending on the value of the DBL bit (page 23), the LANai 5 LBUS operates at either one or two times the chip-clock speed (one or two LBUS memory cycles for every clock cycle). The external-access bus (EBUS), the packet-interface receive DMA, and the packet-interface send DMA each request a maximum of one memory access per clock cycle. The on-chip processor requests up to two memory accesses per clock cycle (instruction and data). The available memory cycle(s) within each clock cycle are assigned based on the following priority (highest to lowest): EBUS, receive DMA, send DMA, and the processor. Since every EBUS memory request is granted, the LANai 5 chip along with the memory on its LBUS appears as a block of synchronous memory when observed from the EBUS.

Both the LBUS and the EBUS addresses are byte addresses, and the byte order is big-endian (the most-significant byte is stored at the lowest byte address).

The 2-, 4-, and 8-byte memory accesses on the LBUS must be aligned; any least-significant bits of an address that would make a memory access non-aligned are ignored.

The LANai chip provides a rudimentary memory-protection mechanism that allows a memory segment of programmable size to be write-protected from the LANai core, i.e., writable only from the EBUS (page 6).

Although the LANai core cannot access the EBUS directly, the on-chip processor can initiate a data transfer between the LBUS and the EBUS (page 5). The LANai EBUS interface is simple and generic, and extra hardware is necessary to connect it to any standard bus.

PACKET SENDING

Please consult pages 13 through 19 for a set of send and receive examples.

A data-communication, flow-control unit is called a flit, and consists of eight data bits plus a tail bit. Packets are of arbitrary length (in flits), and the tail bit marks the last flit of every packet. The byte order in the communication network is big-endian, i.e., the most-significant byte appears first in the network.

After the LANai 5 chip is out of reset, and prior to any Myrinet-network access, the TIMEOUT, MYRINET, and WINDOW special registers (page 7) must be initialized.

Packets are injected into the Myrinet network by initiating the send DMA. The following 32-bit, special registers control the send DMA:

Register	Description
SMP	**Send-Message Pointer:** Address of the first double-word of the send-DMA memory buffer. This register is incremented by 8 by the packet interface as each double-word is appended to the outgoing packet, and, upon completion, equals SML+8.
SA	**Send-Message Align:** The three least-significant bits of this register specify how many leading flits (0-7) of the contents of the next-specified send-DMA memory buffer should NOT be appended to the outgoing packet. This register may be used to keep the payload portion of the message 8-byte aligned, even when the length of the routing header is not a multiple of 8. Only the first send DMA following a write into this register is affected.
SMH	**Send-Message Header:** Address of the last double-word of the routing header. When the CRC-32 is enabled (page 7), writing SMH instructs the packet interface NOT to include the routing header in the CRC-32 for that packet (the routing header will be stripped by the Myrinet switches). Only the first send DMA following a write into this register is affected.
SMC	**Send-Message Header CRC:** Address of the double-word in the send-DMA memory buffer which is NOT appended to the outgoing packet; the optional, partial CRC-32, followed by four zero bytes, is sent instead. Only the first send DMA following a write into this register is affected.
SML	**Send-Message Limit:** Writing this register initiates a send DMA that appends to the outgoing packet, one double-word at a time, the contents of the memory buffer starting with the double-word at address SMP (except for the leading bytes, if specified by SA) and ending with the double-word at address SML. If SMH was written, the CRC-32 computation starts with the double-word at address SMH+8. If SMC was written, the partial CRC-32, followed by four zero flits, is sent instead of the double-word pointed to by SMC.
SMLT	**Send-Message Limit, with the Tail:** The same as SML, but, instead of appending the double-word at address SML, the following byte(s) complete the outgoing packet: 1) if the CRC-32 is enabled, the word equal to the CRC-32 of the outgoing packet (not including any partial CRC-32s), and 2a) if the CRC-8 is enabled (page 7), the tail flit equal to the CRC-8 of the outgoing packet (including all CRC-32 words and sets of four padding zero bytes accompanying partial CRC-32s), or 2b) if the CRC-8 is not enabled, a zero tail flit.

Upon completion of a send DMA, the send_int bit of the special register ISR is set (page 10).

Since the send DMA accesses 64 bits at a time, the send memory buffer must be aligned on a double-word boundary. Hence, the three least-significant bits of SMP, SMH, SMC, and SML(T) are hard-wired to zero.

SA is a write-only register, and reading it produces an undefined value. Reading any other send register is a valid operation, but one should note that SML and SMLT are stored in the same physical register. If any of the send registers are written during a send DMA, the resulting behavior is undefined.

PACKET RECEIVING

Please consult pages 13 through 19 for a set of send and receive examples.

A data-communication, flow-control unit is called a flit, and consists of eight data bits plus a tail bit. Packets are of arbitrary length (in flits), and the tail bit marks the last flit of every packet. The byte order in the communication network is big-endian, i.e., the most significant byte of a word (or of a half-word) appears first in the network.

After the LANai 5 chip is out of reset, and prior to any Myrinet-network access, the TIMEOUT, MYRINET, and WINDOW special registers (page 7) must be initialized.

An incoming packet is accepted from the network by initiating the receive DMA. The following 32-bit, special registers control the receive DMA:

Register	Description
RMP	**Receive-Message Pointer:** Address of the first double-word of the receive-DMA memory buffer. This register is incremented by 8 by the packet interface as each double-word is written into the buffer. After an entire packet has been received, RMP points to the first aligned double-word past the end of the packet.
RMC	**Receive-Message Header CRC:** Points to the slot in the memory for the optional, partial CRC-32. When the CRC-32 is enabled (page 7) and RMC is written, if the message arrives with the correct partial CRC-32, zero is written into the double-word pointed to by RMC. Only the first receive DMA following a write into this register is affected. When the receive DMA has written the incoming message into the memory up to and including the double-word pointed to by RMC, the head_int bit of ISR is set (page 10).
RMW	**Receive-Message Header Wakeup:** This is the same physical register as RMC. However, when writing to RMW there is no side effect of requesting that the partial CRC-32 be verified. The intended use of RMW is to be able to set up an early warning of an incoming message, while the receive DMA is possibly still going on, even when the message header does not carry the partial CRC-32. When the receive DMA has written the incoming message into the memory up to and including the double-word pointed to by RMW, the head_int bit of ISR is set (page 10).
RML	**Receive-Message Limit:** Writing into RML enables a receive DMA and instructs the packet interface to put the (remainder of the) incoming packet, one double-word at a time, into the memory buffer that starts at RMP and ends at RML. When the CRC-8 is enabled (page 7), if the message arrives with the correct CRC-8, zero is written into the last byte of the message. When the CRC-32 is enabled, if the message arrives with the correct CRC-32, zero is written into the four bytes preceding the tail byte.

When an entire incoming packet has been transferred into the receive memory buffer, the recv_int bit of the special register ISR is set (page 10). If the receive memory buffer has been exhausted (the last double-word written is at the location pointed to by RML, and RMP=RML+8), buff_int bit of ISR is set. After a receive DMA is initiated, one must not initiate another receive DMA until the recv_int bit, the buff_int bit, or both, have been set.

Since the receive DMA accesses 64 bits at a time, the receive memory buffer must be aligned on a double-word boundary. Hence, the three least-significant bits of RML, RMP, and RMC (RMW) are hard-wired to zero.

If the length in bytes (flits) of the incoming packet (including any CRC-32 and/or CRC-8) is not a multiple of 8, the overrun bits of ISR will be set (page 10). The packet following the currently accessed packet is guaranteed not to be corrupted. The bytes corresponding to the flits past the tail flit are undefined.

Reading any receive register is a valid operation. If any of these registers is written during a receive DMA, the resulting behavior is undefined.

EBUS-LBUS DATA TRANSFER

The LANai 5 programmer has control over two independent DMA engines, one for EBUS→LBUS direction, the other for LBUS→EBUS direction. The data transfers are initiated by accessing the following 32-bit, special registers:

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>E2L_LAR</td><td><b>LBUS Address Register, EBUS→LBUS Direction:</b> Points to the beginning of the DMA buffer on the LBUS. This register is incremented by 8 as each double-word is transferred, and, upon completion, points to the first double-word past the LBUS DMA buffer.</td></tr>
<tr><td>E2L_EAR</td><td><b>EBUS Address Register, EBUS→LBUS Direction:</b> Points to the beginning of the DMA buffer on the EBUS. This register is incremented by 8 as each double-word is transferred, and, upon completion, points to the first double-word past the EBUS DMA buffer.</td></tr>
<tr><td>E2L_CTR</td><td><b>DMA Counter, EBUS→LBUS Direction:</b> Writing a non-zero value into the E2L_CTR register initiates the EBUS→LBUS DMA. This register is decremented by 8 as each double-word is transferred, and equals 0 upon completion.</td></tr>
<tr><td>L2E_LAR</td><td><b>LBUS Address Register, LBUS→EBUS Direction:</b> Points to the beginning of the DMA buffer on the LBUS. This register is incremented by 8 as each double-word is transferred, and, upon completion, points to the first double-word past the LBUS DMA buffer.</td></tr>
<tr><td>L2E_EAR</td><td><b>EBUS Address Register, LBUS→EBUS Direction:</b> Points to the beginning of the DMA buffer on the EBUS. This register is incremented by 8 as each double-word is transferred, and, upon completion, points to the first double-word past the EBUS DMA buffer.</td></tr>
<tr><td>L2E_CTR</td><td><b>DMA Counter, LBUS→EBUS Direction:</b> Writing a non-zero value into the E2L_CTR register initiates the LBUS→EBUS DMA. This register is decremented by 8 as each double-word is transferred, and equals 0 upon completion.</td></tr>
<tr><td>LAR</td><td><b>LBUS Address Register:</b> Reserved for use by the EBUS hardware (page 28).</td></tr>
<tr><td>CTR</td><td><b>EBUS-LBUS DMA Counter:</b> Reserved for use by the EBUS hardware (page 28).</td></tr>
<tr><td>BURST</td><td><b>EBUS-LBUS DMA Burst Sizes:</b> The eight least-significant bits of this register specify the allowed burst modes of the EBUS interface. The BURST register affects only the LANai 5 status pins (page 37).
<table>
<tr><th>Bit</th><th>Name</th><th>PCI</th><th>SBUS</th></tr>
<tr><td>0</td><td>PCI</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>B16</td><td>16-byte cache-line size</td><td>16-byte bursts supported</td></tr>
<tr><td>2</td><td>B32</td><td>32-byte cache-line size</td><td>32-byte bursts supported</td></tr>
<tr><td>3</td><td>B64</td><td>64-byte cache-line size</td><td>64-byte bursts supported</td></tr>
<tr><td>4</td><td>B128</td><td>128-byte cache-line size</td><td>128-byte bursts supported</td></tr>
<tr><td>5</td><td>B256</td><td>256-byte cache-line size</td><td>N/A</td></tr>
<tr><td>6</td><td>B512</td><td>512-byte cache-line size</td><td>N/A</td></tr>
<tr><td>7</td><td>B1024</td><td>1024-byte cache-line size</td><td>N/A</td></tr>
</table>
For PCI-like interfaces, which support only a single burst size, typically equal to the cache-line size, only one of the bits 1 through 7 should be set. For SBUS-like interfaces, which support several burst sizes, all the supported burst sizes should have their corresponding bits set.</td></tr>
</table>

Upon completion of an EBUS→LBUS DMA the e2l_int bit of the special register ISR is set (page 10). Upon completion of an LBUS→EBUS DMA, the l2e_int bit of ISR is set.

Since the EBUS DMAs transfer 64 bits at a time, the LBUS memory buffer must be aligned on a double-word boundary. Hence, the three least-significant bits of E2L_LAR, E2L_CTR, L2E_LAR, and L2E_CTR are hard-wired to zero.

The BURST is a write-only register, and reading it produces undefined values. Reading any other register is a valid operation. Writing any of the E2L registers or the BURST register during an EBUS→LBUS DMA, or writing any of the L2E registers or the BURST register during an LBUS→EBUS DMA may result in violation of the protocol on the I/O bus that the EBUS connects to.

COUNTERS/TIMERS

There are two real-time counters on the LANai 5 chip, both of which use the time reference that is equal to 40 times the period of the transmit clock of the Myrinet-SAN interface (page 39). Nominally, this is an 80 MHz clock, so the time reference is equal to 1/2 microsecond.

Register	Description
RTC	**Real-Time Clock:** This is a 32-bit counter that is incremented every time-reference period.
IT	**Interrupt Timer:** This is a 32-bit counter that is decremented every time-reference period. Whenever this counter makes a transition from 0x00000000 to 0xFFFFFFFF, the time_int bit of the special register ISR is set (page 10). Whenever it makes a transition from 0x80000000 to 0x7FFFFFFF, the wdog_int bit of ISR is set.

MEMORY PROTECTION

The LANai 5 chip provides a rudimentary memory-protection mechanism that allows a memory segment of programmable size to be write-protected from the LANai core, i.e., writable only from the EBUS.

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>MP</td><td><b>Memory Protection:</b> If the WE (Write Enable) bit is 1, the LANai is allowed to write to any memory location (no memory protection). Upon reset, this is the default value of the WE bit.
If the WE bit is 0, the A12-A23 bits (the A bits) define the region(s) of memory in which the LANai core is allowed to write: a write to a memory location is allowed if a bit in the address of that memory location is 1 and the corresponding A bit is 1.
<table>
<tr><td><b>Bit</b></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td></tr>
<tr><td><b>Name</b></td><td>WE</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td><b>Bit</b></td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr>
<tr><td><b>Name</b></td><td>A23</td><td>A22</td><td>A21</td><td>A20</td><td>A19</td><td>A18</td><td>A17</td><td>A16</td></tr>
<tr><td><b>Bit</b></td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td></tr>
<tr><td><b>Name</b></td><td>A15</td><td>A14</td><td>A13</td><td>A12</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td><b>Bit</b></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td><b>Name</b></td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
</table>
</td></tr>
</table>

A typical use of this mechanism is to write-protect a memory segment at the bottom of the memory (where the code for the LANai processor is usually kept), and allow writes to addresses up to the highest available memory on the LBUS.

For example, writing the value 0x000FE000 to the MP special register write-protects the lowest 8KB and allows writes to addresses up to 1MB-1.

Pin-count limitations restrict the LBUS of the LANai 5 chip to a maximum of 16M bytes.

The MP special register is a write-only register, and reading it produces an undefined value.

CONFIGURATION

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>TIMEOUT</td><td><b>Incoming-Message Blocking Timeout:</b> the two least-significant bits of the TIMEOUT special register specify the timeout period of the LANai watchdog timer. If the LANai 5 chip fails to consume an incoming message from the network for the duration of the timeout period, the watchdog timer sets the nres_int bit of the special register ISR (page 10), and, if the NRES_ENABLE bit of the special register MYRINET is set, it resets the LANai chip.
<table>
<tr><th>Value</th><th>Timeout Period</th></tr>
<tr><td>0</td><td>1/16 second</td></tr>
<tr><td>1</td><td>1/4 second</td></tr>
<tr><td>2</td><td>1 second</td></tr>
<tr><td>3</td><td>4 seconds</td></tr>
</table></td></tr>
<tr><td>WINDOW</td><td><b>SAN-Link Sampling Window:</b> The two least-significant bits of this register select the width of the character-time window for the input section of the Myrinet-SAN interface. After the chip is out of reset and prior to any Myrinet access, the system-specific value (page 12) must be written to this register, and a minimum of 10 milliseconds must be allowed for the SAN link to reconfigure.</td></tr>
<tr><td>MYRINET</td><td><b>Myrinet-Link Configuration:</b> The three least-significant bits of this register enable the error-handling features of the Myrinet-SAN interface. After the chip is out of reset, this register must be written prior to any Myrinet access.
<table>
<tr><th>Bit</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>NRES_ENABLE</td><td>When the LANai 5 chip fails to consume an incoming message for the duration of the period selected by the TIMEOUT register, the nres_int bit of the special register ISR is set (page 10). If the NRES_ENABLE is set, the chip will be reset when the nres_int bit is set.</td></tr>
<tr><td>1</td><td>CRC8_ENABLE</td><td>Enables the CRC-8 computation.</td></tr>
<tr><td>2</td><td>CRC32_ENABLE</td><td>Enables the CRC-32 computation.</td></tr>
</table></td></tr>
<tr><td>DEBUG</td><td><b>Hardware-Debug Register:</b> The five least-significant bits of this register select one of 32 internal signals to be output on the WIN pin, for timing observation.</td></tr>
<tr><td>CLOCK</td><td><b>Internal-Clock Phase-Adjusting Register:</b> This special register controls the on-chip clock generation. During the power-on reset, the system-specific value must be written to this register from the EBUS (page 12). This register may be modified only while the chip is in reset (page 25).</td></tr>
</table>

PROGRAMMABLE OUTPUTS

Register	Description
LED	**LED Register:** The least-significant bit of this special register is driven to the LED output pin.
PULSE	**PULSE Register:** The three least-significant bits of this special register correspond to the three output pins: P0, P1, and P2. When a value of 1 is written into such a bit, a one-clock-cycle-long pulse is generated on the corresponding output pin.

The special registers described on this page are write-only registers, and reading any of them produces an undefined value.

Note: There is no PULSE register in the LANai5.0 version of the chip.

INTERRUPTS

Register	Description
ISR	**Interrupt Status Register:** Contains the chip-status information. The ISR bits with the _sig (signal) postfix are included for simple host-LANai communication. A signal bit can be set only by the LANai processor and reset only from the EBUS, or vice versa. The ISR bits with the _int (interrupt) postfix are set by the packet interface or the EBUS interface when their corresponding events occur. These bits can be reset directly — by writing a 1 into them, or indirectly — in a bit-specific way.
IMR	**Interrupt Mask Register:** When a bit of ISR is equal to 1 and the corresponding bit of IMR is equal to 1, an interrupt request is asserted for the on-chip processor.
EIMR	**External-Interrupt Mask Register:** When a bit of ISR is equal to 1 and the corresponding bit of EIMR is equal to 1, the INT output pin is asserted.

Accessing some special registers has a side effect of clearing ISR bits. There is up to one-assembly-instruction delay between the time when a special register is accessed and the time of the clearing of the corresponding ISR bit(s). In case of tight polling loops, this delay can result in a race condition, whereby the program could, for example, misinterpret a not-yet-cleared ISR _int bit for an indication of a new event. The following table specifies which special-register write clears which ISR bits:

Writing	Clears
RML	buff_int head_int orun4_int orun2_int orun1_int recv_int
SML, SMLT	send_int
IT	time_int wdog_int
E2L_CTR	e2l_int
L2E_CTR	l2e_int

The ISR, IMR, and EIMR consist of the following bits (bit 31 is the most significant):

Bit	Name	Description
31	debug_bit	This bit is always equal to 1, and can be used for single-stepping the code that runs in user context (page ??).
30	host_sig	This bit is set when the LANai processor writes a 1 into it, and reset when a 1 is written into it from the EBUS.
29-24	0	Reserved.
23-16	lan7_sig - lan0_sig	Each of these 8 bits is set when a 1 is written into it from the EBUS, and reset when the LANai processor writes a 1 into it.

Bit	Name	Description
15-13	0	Reserved.
12	wake_int	This bit is set when the WAKE input pin is asserted. This bit is cleared by the programmer, only directly — by writing a 1 into it.
11	nres_int	This bit is set by the Myrinet-SAN interface whenever the LANai chip fails to consume an incoming message from the Myrinet network for the duration of the period specified by the TIMEOUT special register. If the NRES_ENABLE bit of the MYRINET special register is 1, the LANai chip is also reset. By examining the nres_int bit, one can distinguish between the reset-pin-induced and NRES-induced reset. This bit is cleared by the programmer, only directly — by writing a 1 into it.
10 - 8	orun4_int orun2_int orun1_int	These bits are set by the packet interface when an overrun condition is detected, i.e., when the length in bytes (flits) of the incoming packet (including any CRC-32 and/or CRC-8) is not a multiple of 8. The three bits taken together represent the number of bytes in the receive buffer beyond the tail byte (0 through 7). For example, if the tail is received in the most-significant byte of a double-word, all three bits will be set, indicating that the values of the 7 least-significant bytes are undefined. These bits are cleared by the programmer, either directly — by writing a 1 into them, or indirectly — when RML is written.
7	wdog_int	This bit is set by the interrupt timer whenever it makes a transition from 0x80000000 to 0x7FFFFFFF. This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when IT is written.
6	time_int	This bit is set by the interrupt timer whenever it makes a transition from 0x00000000 to 0xFFFFFFFF. This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when IT is written.
5	l2e_int	This bit is set by the LBUS→EBUS DMA engine when the L2E_CTR reaches 0 to signal the completion of a DMA. This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when the L2E_CTR is written. After an LBUS→EBUS DMA is initiated, one must not initiate another such DMA until the l2e_int bit becomes 1.
4	e2l_int	This bit is set by the EBUS→LBUS DMA engine when the E2L_CTR reaches 0 to signal the completion of a DMA. This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when the E2L_CTR is written. After an EBUS→LBUS DMA is initiated, one must not initiate another such DMA until the e2l_int bit becomes 1.
3	send_int	This bit is set by the packet interface to signal the completion of a send DMA, i.e., when the contents of the send memory buffer and any associated CRCs have been appended to the outgoing packet. This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when SML(T) is written. After a send DMA is initiated, one must not initiate another send DMA until the send_int bit becomes 1.
2	buff_int	This bit is set by the packet interface when the receive-DMA buffer has been exhausted (the last double-word written is at the location pointed to by RML, and RMP=RML+8). This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when RML is written. After a receive DMA is initiated, one must not initiate another receive DMA until the recv_int bit, the buff_int bit, or both, become 1.
1	recv_int	This bit is set by the packet interface to signal the completion of a receive DMA, i.e., when the entire incoming packet has been transferred into the receive memory buffer. This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when RML is written. After a receive DMA is initiated, one must not initiate another receive DMA until the recv_int bit, the buff_int bit, or both, become 1.
0	head_int	This bit is set by the packet interface to signal that the head of a packet (up to and including the double-word pointed to by RMW (RMC)) has been received into the memory. The receive DMA continues until: 1) an entire incoming packet has been transferred into the receive memory buffer, or 2) the receive-DMA buffer has been exhausted. This bit is cleared by the programmer, either directly — by writing a 1 into it, or indirectly — when RML is written.

SPECIAL-REGISTER SUMMARY

Register	Read		Write		Description	Offset	Page
	EBUS	LANai	EBUS	LANai			
BURST			+	+	EBUS-DMA Configuration	0x120	5
CLOCK			+		Clock Configuration	0x1F8	7
CTR	+	+	+	+	EBUS-DMA Counter	0x78	5
DEBUG			+	+	Hardware Debugging	0x138	7
E2L_CTR	+	+	+	+	Initiate E→L DMA	0xA8	5
E2L_EAR	+	+	+	+	E→L EBUS Address	0x98	5
E2L_LAR	+	+	+	+	E→L LBUS Address	0x88	5
EIMR	+	+	+	+	External-Interrupt Mask	0x58	9
IMR		+		+	LANai-Interrupt Mask	-	9
ISR	+	+	+	+	Interrupt Status	0x50	9
IT	+	+	+	+	Interrupt Timer	0x60	6
L2E_CTR	+	+	+	+	Initiate L→E DMA	0xA0	5
L2E_EAR	+	+	+	+	L→E EBUS Address	0x90	5
L2E_LAR	+	+	+	+	L→E LBUS Address	0x80	5
LAR	+	+	+	+	EBUS-DMA LBUS address	0x70	5
LED			+	+	LED Output Pin	0x140	8
PULSE			+	+	P0, P1, P2 Output Pins	0xB8	8
MP			+	+	Memory Protection	0x150	6
MYRINET			+	+	Myrinet-Link Configuration	0x130	7
RMC	+	+	+	+	Receive-DMA Header CRC	0xD8	4
RML	+	+	+	+	Initiate Receive DMA	0xE8	4
RMP	+	+	+	+	Receive-DMA Buffer	0xE0	4
RMW	+	+	+	+	Receive-DMA Header	0xD0	4
RTC	+	+	+	+	Real-Time Clock	0x68	6
SA			+	+	Send-DMA Alignment	0x118	3
SMC	+	+	+	+	Send-DMA Header CRC	0x110	3
SMH	+	+	+	+	Send-DMA Routing Header	0xF8	3
SML	+	+	+	+	Initiate Send DMA	0x100	3
SMLT	+	+	+	+	Initiate Send DMA with Tail	0x108	3
SMP	+	+	+	+	Send-DMA Buffer	0xF0	3
TIMEOUT			+	+	NRES-Timeout Selection	0x128	7
WINDOW			+	+	Sampling-Window Selection	0x148	7

All special registers except for the IMR are memory-mapped. The IMR is an internal register of the LANai on-chip processor and is not accessible from the EBUS. The memory-mapped special registers can be accessed both by the LANai on-chip processor and from the EBUS (except for the special register CLOCK, that can be accessed only from the EBUS).

To access a memory-mapped special register from the LANai processor one should use the address of 0xFFFFFE00 plus the offset of that special register. The base address for EBUS access of memory-mapped special registers is application-specific; consult system documentation for details.

When accessing the special, memory-mapped registers, the regular memory arbitration mechanism described on page 2 applies. The mutual exclusion at any higher level is the responsibility of the programmer.

INITIALIZATION

During the power-on reset, the system-specific value listed below must be written to the CLOCK special register from the EBUS, to initialize the on-chip clock generation, and the chip must be held in reset a minimum of 10 milliseconds to allow the on-chip PLL to stabilize.

After chip reset, the on-chip processor begins executing code in the system context, starting from the address 0.

The state of the nres_int bit in the ISR upon reset indicates whether the reset has been a regular, reset-pin initialization (0), or an NRES-induced reset (1).

All the remaining bits of ISR are equal to 0, except the debug_bit, which is equal to 1.

The MP register is initialized to the no-memory-protection state.

The IMR and the EIMR special registers are undefined and should be initialized by the programmer.

All other special registers are initialized to 0 upon reset.

After the chip is out of reset and prior to any Myrinet access, the system-specific value listed below must be written to the WINDOW register to configure the Myrinet-SAN interface, and a minimum of 10 milliseconds must be allowed for the SAN link to reconfigure.

SYSTEM-SPECIFIC INITIALIZATION

Version	CLOCK	WINDOW
LANai5.0	0x06698200	3
LANai5.2 40MHz CLK 10ns LBUS SRAM	0x08298281	3

SIMPLE MESSAGE SENDING AND RECEIVING EXAMPLES

In all the examples, data is represented by the following symbols:

```
        'm'   -  message byte, used in CRC-32 computation
        'h'   -  header byte, not used in CRC-32 computation
        '.'   -  don't-care byte
        'c8'  -  CRC-8 byte
        'c32' -  one of four CRC-32 bytes
        'v8'  -  a verified CRC-8 byte (zero if CRC-8 caught no errors)
        'v32' -  one of four verified CRC-32 bytes (zero if CRC-32 caught no errors)
```

Example #1:

```
      CRC-32  off
      CRC-8   off
                                                    sender memory
                                             msb                             lsb
sender ()                                    +---+---+---+---+---+---+---+---+
{                                   SMP ->   | m | m | m | m | m | m | m | m |
  SMP  = 0x1000;                   (start)   +---+---+---+---+---+---+---+---+
  SMLT = 0x1010;                             | m | m | m | m | m | m | m | m |
}                                            +---+---+---+---+---+---+---+---+
                                    SMLT ->  | . | . | . | . | . | . | . | . |
                                             +---+---+---+---+---+---+---+---+
                                    SMP ->   | . | . | . | . | . | . | . | . |
                                    (end)    +---+---+---+---+---+---+---+---+


                                                   bytes in Myrinet
                                             +---+---+---+---+---+---+---+---+
                                             | m | m | m | m | m | m | m | m |
                                             +---+---+---+---+---+---+---+---+
                                             | m | m | m | m | m | m | m | m |
                                             +---+---+---+---+---+---+---+---+
                                             | 0 |
                                             +---+

                                                   receiver memory
                                             msb                             lsb
receiver ()                                  +---+---+---+---+---+---+---+---+
{                                   RMP ->   | m | m | m | m | m | m | m | m |
  RMP = 0x2000;                    (start)   +---+---+---+---+---+---+---+---+
  RML = 0x3000;                              | m | m | m | m | m | m | m | m |
}                                            +---+---+---+---+---+---+---+---+
                                             | 0 | . | . | . | . | . | . | . |
                                             +---+---+---+---+---+---+---+---+
                                    RMP ->   | . | . | . | . | . | . | . | . |
                                    (end)    +---+---+---+---+---+---+---+---+
                                               .   .   .   .   .   .   .   .
                                               .   .   .   .   .   .   .   .
                                               .   .   .   .   .   .   .   .
                                             +---+---+---+---+---+---+---+---+
                                    RML ->   | . | . | . | . | . | . | . | . |
                                             +---+---+---+---+---+---+---+---+
```

Example #2:

```
        CRC-32   off
        CRC-8    on
                                                  sender memory
                                        msb                           lsb
sender ()                               +---+---+---+---+---+---+---+---+
{                               SMP ->  | m | m | m | m | m | m | m | m |
  SMP  = 0x1000;              (start)   +---+---+---+---+---+---+---+---+
  SMLT = 0x1010;                        | m | m | m | m | m | m | m | m |
}                                       +---+---+---+---+---+---+---+---+
                               SMLT ->  | . | . | . | . | . | . | . | . |
                                        +---+---+---+---+---+---+---+---+
                                SMP ->  | . | . | . | . | . | . | . | . |
                               (end)    +---+---+---+---+---+---+---+---+


                                                 bytes in Myrinet
                                        +---+---+---+---+---+---+---+---+
                                        | m | m | m | m | m | m | m | m |
                                        +---+---+---+---+---+---+---+---+
                                        | m | m | m | m | m | m | m | m |
                                        +---+---+---+---+---+---+---+---+
                                        | c8|
                                        +---+


                                                 receiver memory
                                        msb                           lsb
                                        +---+---+---+---+---+---+---+---+
receiver ()                     RMP ->  | m | m | m | m | m | m | m | m |
{                             (start)   +---+---+---+---+---+---+---+---+
  RMP = 0x2000;                         | m | m | m | m | m | m | m | m |
  RML = 0x3000;                         +---+---+---+---+---+---+---+---+
}                                       | v8| . | . | . | . | . | . | . |
                                        +---+---+---+---+---+---+---+---+
                                RMP ->  | . | . | . | . | . | . | . | . |
                               (end)    +---+---+---+---+---+---+---+---+
                                          .   .   .   .   .   .   .   .
                                          .   .   .   .   .   .   .   .
                                          .   .   .   .   .   .   .   .
                                        +---+---+---+---+---+---+---+---+
                                RML ->  | . | . | . | . | . | . | . | . |
                                        +---+---+---+---+---+---+---+---+
```

Example #3:

```
        CRC-32   on
        CRC-8    on
```

```
sender ()
{
  SMP  = 0x1000;
  SMLT = 0x1010;
}
```

```
                       sender memory
            msb                             lsb
            +---+---+---+---+---+---+---+---+
  SMP ->    | m | m | m | m | m | m | m | m |
(start)     +---+---+---+---+---+---+---+---+
            | m | m | m | m | m | m | m | m |
            +---+---+---+---+---+---+---+---+
  SMLT ->   | . | . | . | . | . | . | . | . |
            +---+---+---+---+---+---+---+---+
  SMP ->    | . | . | . | . | . | . | . | . |
 (end)      +---+---+---+---+---+---+---+---+
```

```
                    bytes in Myrinet
            +---+---+---+---+---+---+---+---+
            | m | m | m | m | m | m | m | m |
            +---+---+---+---+---+---+---+---+
            | m | m | m | m | m | m | m | m |
            +---+---+---+---+---+---+---+---+
            |c32|c32|c32|c32| c8|
            +---+---+---+---+---+
```

```
receiver ()
{
  RMP = 0x2000;
  RML = 0x3000;
}
```

```
                     receiver memory
            msb                             lsb
            +---+---+---+---+---+---+---+---+
  RMP ->    | m | m | m | m | m | m | m | m |
(start)     +---+---+---+---+---+---+---+---+
            | m | m | m | m | m | m | m | m |
            +---+---+---+---+---+---+---+---+
            |v32|v32|v32|v32| v8| . | . | . |
            +---+---+---+---+---+---+---+---+
  RMP ->    | . | . | . | . | . | . | . | . |
 (end)      +---+---+---+---+---+---+---+---+
              .   .   .   .   .   .   .   .
              .   .   .   .   .   .   .   .
              .   .   .   .   .   .   .   .
            +---+---+---+---+---+---+---+---+
  RML ->    | . | . | . | . | . | . | . | . |
            +---+---+---+---+---+---+---+---+
```

Example #4:

```
        CRC-32   on
        CRC-8    on
        SMH      used
```

```
                                                  sender memory
                                         msb                             lsb
sender ()                                +---+---+---+---+---+---+---+---+
{                               SMP ->   | h | h | h | h | h | h | h | h |
  SMP  = 0x1000;              (start)    +---+---+---+---+---+---+---+---+
  SMH  = 0x1008;                SMH ->   | h | h | h | h | h | h | h | h |
  SMLT = 0x1020;                         +---+---+---+---+---+---+---+---+
}                                        | m | m | m | m | m | m | m | m |
                                         +---+---+---+---+---+---+---+---+
                                         | m | m | m | m | m | m | m | m |
                                         +---+---+---+---+---+---+---+---+
                                SMLT ->  | . | . | . | . | . | . | . | . |
                                         +---+---+---+---+---+---+---+---+
                                SMP ->   | . | . | . | . | . | . | . | . |
                                 (end)   +---+---+---+---+---+---+---+---+
```

```
                                                 bytes in Myrinet
                                         +---+---+---+---+---+---+---+---+
                                         | h | h | h | h | h | h | h | h |
                                         +---+---+---+---+---+---+---+---+
                                         | h | h | h | h | h | h | h | h |
                                         +---+---+---+---+---+---+---+---+
                                         | m | m | m | m | m | m | m | m |
                                         +---+---+---+---+---+---+---+---+
                                         | m | m | m | m | m | m | m | m |
                                         +---+---+---+---+---+---+---+---+
                                         |c32|c32|c32|c32| c8|
                                         +---+---+---+---+---+
```

```
                                                 receiver memory
                                         (assuming the first 16 bytes
                                          have been stripped by switches)
                                         msb                             lsb
receiver ()                              +---+---+---+---+---+---+---+---+
{                               RMP ->   | m | m | m | m | m | m | m | m |
  RMP = 0x2000;               (start)    +---+---+---+---+---+---+---+---+
  RML = 0x3000;                          | m | m | m | m | m | m | m | m |
}                                        +---+---+---+---+---+---+---+---+
                                         |v32|v32|v32|v32| v8| . | . | . |
                                         +---+---+---+---+---+---+---+---+
                                RMP ->   | . | . | . | . | . | . | . | . |
                                 (end)   +---+---+---+---+---+---+---+---+
                                           .   .   .   .   .   .   .   .
                                           .   .   .   .   .   .   .   .
                                           .   .   .   .   .   .   .   .
                                         +---+---+---+---+---+---+---+---+
                                RML ->   | . | . | . | . | . | . | . | . |
                                         +---+---+---+---+---+---+---+---+
```

Example #5:

```
        CRC-32    on
        CRC-8     on
        SMH       used
        SA        used
```

```
                                           sender memory
                                    msb                             lsb
sender ()                           +---+---+---+---+---+---+---+---+
{                          SMP ->   | . | . | . | . | . | h | h | h |
  SMP  = 0x1000;         (start)    +---+---+---+---+---+---+---+---+
  SMH  = 0x1008;           SMH ->   | h | h | h | h | h | h | h | h |
  SA   = 0x5;                       +---+---+---+---+---+---+---+---+
  SMLT = 0x1020;                    | m | m | m | m | m | m | m | m |
}                                   +---+---+---+---+---+---+---+---+
                                    | m | m | m | m | m | m | m | m |
                                    +---+---+---+---+---+---+---+---+
                           SMLT ->  | . | . | . | . | . | . | . | . |
                                    +---+---+---+---+---+---+---+---+
                           SMP ->   | . | . | . | . | . | . | . | . |
                          (end)     +---+---+---+---+---+---+---+---+
```

```
                                          bytes in Myrinet
                                                        +---+---+---+
                                                        | h | h | h |
                                    +---+---+---+---+---+---+---+---+
                                    | h | h | h | h | h | h | h | h |
                                    +---+---+---+---+---+---+---+---+
                                    | m | m | m | m | m | m | m | m |
                                    +---+---+---+---+---+---+---+---+
                                    | m | m | m | m | m | m | m | m |
                                    +---+---+---+---+---+---+---+---+
                                    |c32|c32|c32|c32| c8|
                                    +---+---+---+---+---+
```

```
                                           receiver memory
                                    (assuming the first 11 bytes
                                     have been stripped by switches)
                                    msb                             lsb
receiver ()                         +---+---+---+---+---+---+---+---+
{                          RMP ->   | m | m | m | m | m | m | m | m |
  RMP = 0x2000;          (start)    +---+---+---+---+---+---+---+---+
  RML = 0x3000;                     | m | m | m | m | m | m | m | m |
}                                   +---+---+---+---+---+---+---+---+
                                    |v32|v32|v32|v32| v8| . | . | . |
                                    +---+---+---+---+---+---+---+---+
                           RMP ->   | . | . | . | . | . | . | . | . |
                          (end)     +---+---+---+---+---+---+---+---+
                                      .   .   .   .   .   .   .   .
                                      .   .   .   .   .   .   .   .
                                      .   .   .   .   .   .   .   .
                                    +---+---+---+---+---+---+---+---+
                           RML ->   | . | . | . | . | . | . | . | . |
                                    +---+---+---+---+---+---+---+---+
Example #6:

        CRC-32   on
        CRC-8    on
        SMH      used
        SA       used
        SMC      used
        RMC      used
                                                sender memory
                                        msb                              lsb
sender ()                               +---+---+---+---+---+---+---+---+
{                               SMP ->  | . | . | . | . | . | h | h | h |
  SMP  = 0x1000;              (start)   +---+---+---+---+---+---+---+---+
  SMH  = 0x1008;                SMH ->  | h | h | h | h | h | h | h | h |
  SA   = 0x5;                           +---+---+---+---+---+---+---+---+
  SMC  = 0x1020;                        | m | m | m | m | m | m | m | m |
  SMLT = 0x1038;                        +---+---+---+---+---+---+---+---+
}                                       | m | m | m | m | m | m | m | m |
                                        +---+---+---+---+---+---+---+---+
                                SMC ->  | . | . | . | . | . | . | . | . |
                                        +---+---+---+---+---+---+---+---+
                                        | m | m | m | m | m | m | m | m |
                                        +---+---+---+---+---+---+---+---+
                                        | m | m | m | m | m | m | m | m |
                                        +---+---+---+---+---+---+---+---+
                                SMLT -> | . | . | . | . | . | . | . | . |
                                        +---+---+---+---+---+---+---+---+
                                SMP ->  | . | . | . | . | . | . | . | . |
                                (end)   +---+---+---+---+---+---+---+---+


                                               bytes in Myrinet
                                                            +---+---+---+
                                                            | h | h | h |
                                        +---+---+---+---+---+---+---+---+
                                        | h | h | h | h | h | h | h | h |
                                        +---+---+---+---+---+---+---+---+
                                        | m | m | m | m | m | m | m | m |
                                        +---+---+---+---+---+---+---+---+
                                        | m | m | m | m | m | m | m | m |
                                        +---+---+---+---+---+---+---+---+
                    partial CRC-32 ->   |c32|c32|c32|c32| 0 | 0 | 0 | 0 |
                    (on m's up to here) +---+---+---+---+---+---+---+---+
                                        | m | m | m | m | m | m | m | m |
                                        +---+---+---+---+---+---+---+---+
                                        | m | m | m | m | m | m | m | m |
                                        +---+---+---+---+---+---+---+---+
                    full CRC-32 ->      |c32|c32|c32|c32| c8|
                    (on m's up to here) +---+---+---+---+---+
receiver ()
{
  RMP = 0x2000;
  RMC = 0x2010;
  RML = 0x3000;
}
```

```
                      receiver memory
               (assuming the first 11 bytes
                have been stripped by switches)
               msb                            lsb
               +---+---+---+---+---+---+---+---+
     RMP ->    | m | m | m | m | m | m | m | m |
    (start)    +---+---+---+---+---+---+---+---+
               | m | m | m | m | m | m | m | m |
               +---+---+---+---+---+---+---+---+
     RMC ->    |v32|v32|v32|v32| 0 | 0 | 0 | 0 |
               +---+---+---+---+---+---+---+---+
               | m | m | m | m | m | m | m | m |
               +---+---+---+---+---+---+---+---+
               | m | m | m | m | m | m | m | m |
               +---+---+---+---+---+---+---+---+
               |v32|v32|v32|v32| v8| . | . | . |
               +---+---+---+---+---+---+---+---+
     RMP ->    | . | . | . | . | . | . | . | . |
     (end)     +---+---+---+---+---+---+---+---+
                 .   .   .   .   .   .   .   .
                 .   .   .   .   .   .   .   .
                 .   .   .   .   .   .   .   .
               +---+---+---+---+---+---+---+---+
     RML ->    | . | . | . | . | . | . | . | . |
               +---+---+---+---+---+---+---+---+
```

ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS

Symbol	Rating	Value	Unit
V_{dd}	Power Supply Voltage	-0.5 to +4.6	V
V_{in}, V_{out}	Terminal Voltage (except Vdd)	-0.5 to V_{dd}+0.5	V
I_{out}	Output Current	100	mA
P_D	Power Dissipation	3	W
T_{bias}	Temperature Under Bias	-55 to 125	°C
T_{stg}	Storage Temperature	-55 to 125	°C
T_A	Operating Temperature	0 to 70	°C

RECOMMENDED OPERATING CONDITIONS

Symbol	Parameter	Min	Typ	Max	Unit
V_{dd}	Power Supply Voltage	3.0	3.3	3.6	V
V_{IH}	Input High Voltage	2.2	-	Vdd+0.3	V
V_{IL}	Input Low Voltage	-0.3	-	0.8	V

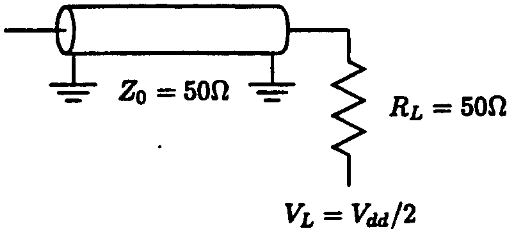
DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Unit
I_{LI}	Input Leakage Current	-	±1.0	μA
I_{LO}	Output Leakage Current	-	±1.0	μA
$V_{OL}(IOL = 5mA)$	Output Low Voltage	-	0.4	V
$V_{OH}(IOH = -5mA)$	Output High Voltage	2.4	-	V

CAPACITANCE ($T_A = +25°C, f = 1.0MHz, dV = 3V$)

Symbol	Parameter	Max	Unit
C_I	Input Capacitance	5	pF
$C_{I/O}$	I/O Capacitance	8	pF

AC TEST LOADS



OUTPUT DRIVERS

The output drivers and the tri-state drivers are of three different strengths. For purely capacitive loads, the following are the typical delay values at $V_{dd} = 3.3V, T_a = 25^oC$:

Output	Delay
A03 - A23, A18, A20, A22, OE, WE0 - WE7	$out_L = 0.2ns + C_{\text{load}}[pF] * 0.015ns/pF$
ED00 - ED63	$out_E = 0.2ns + C_{\text{load}}[pF] * 0.030ns/pF$
otherwise	$out = 0.2ns + C_{\text{load}}[pF] * 0.042ns/pF$

For power-supply voltage of $3.3V \pm 10\%$, ambient temperature from 0^o to 70^oC, and the acceptable manufacturing-process variation, output-driver delays have ±50% variation.

The full hspice model is available upon request.

Note: In the LANai5.0 version of the chip, the $\overline{\text{WE}}$ drivers are weaker, the same strength as the ED drivers shown above.

PINOUT

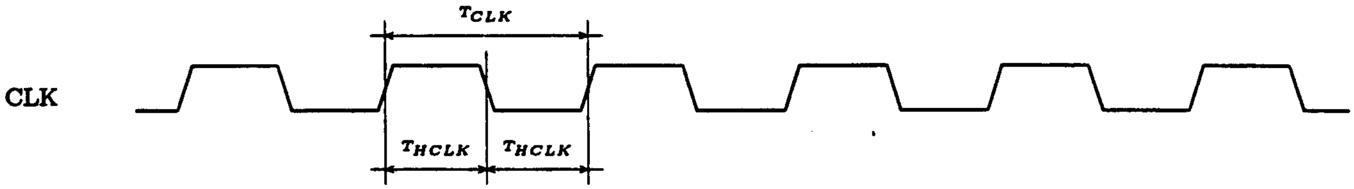
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
A	Vdd	GND	I7	I4	I0	GND	O3	GND	O6	OD	DBL	GND	A04	A08	A11	GND	A17	GND	A22	$\overline{\text{WE2}}$	$\overline{\text{WE5}}$	GND	Vdd
B	GND	Vdd	GND	I6	I3	Vth	LVdd	O4	LVdd	LVdd	OAH	PLL	A05	A09	A13	A15	A18	A21	$\overline{\text{WE1}}$	$\overline{\text{WE4}}$	GND	Vdd	GND
C	$\overline{\text{A22}}$	GND	Vdd	IB	I5	I2	O0	O2	BIAS	O7	OAL	A03	A06	A10	A14	A16	A20	$\overline{\text{WE0}}$	$\overline{\text{WE3}}$	$\overline{\text{WE7}}$	Vdd	GND	D00
D	A23	$\overline{\text{A20}}$	$\overline{\text{RST}}$	Vdd	ID	Vdd	I1	O1	Vdd	O5	OB	Vdd	A07	A12	Vdd	A19	$\overline{\text{OE}}$	Vdd	$\overline{\text{WE6}}$	Vdd	.	D01	D03
E	TCLK	WAKE	$\overline{\text{A18}}$	ERST																SELR	D02	D04	D07
F	GND	WIN	INT	Vdd																Vdd	D05	D08	GND
G	E2LA	SNB	RCLK	LED																D06	D09	D11	D13
H	GND	E2LP	L2EA	CLK																D10	D12	D14	GND
J	EWE7	BUSY	L2EP	Vdd																Vdd	D15	D16	D18
K	EWE4	EWE5	EWE6	FRDY																D17	D19	D20	D21
L	EWE0	EWE1	EWE2	EWE3																D22	D23	D24	D25
M	GND	ED63	.	Vdd																Vdd	D27	D26	GND
N	ED62	ED61	ED60	ED59																D31	D30	D29	D28
P	ED58	ED57	ED56	ED54																D36	D34	D33	D32
R	ED55	ED53	ED52	Vdd																Vdd	D38	D37	D35
T	GND	ED51	ED49	ED47																D43	D40	D39	GND
U	ED50	ED48	ED46	ED43																D47	D44	D42	D41
V	GND	ED45	ED42	Vdd																Vdd	D48	D45	GND
W	ED44	ED41	ED39	WMEM																S8	D51	D49	D46
Y	ED40	DMA	$\overline{\text{EOE}}$	Vdd	RFIFO	Vdd	ED34	ED30	Vdd	ED23	ED18	Vdd	ED09	ED04	Vdd	D61	D57	Vdd	S2	Vdd	S4	D52	D50
AA	RMEM	GND	Vdd	.	ED38	ED35	ED31	ED27	ED25	ED21	ED17	ED14	ED10	ED06	ED02	D63	D60	D56	D53	.	Vdd	GND	S16
AB	GND	Vdd	GND	L2ES	ED36	ED32	ED29	ED26	ED24	ED20	ED16	ED13	ED11	ED07	ED03	ED01	D62	D59	D55	SC	GND	Vdd	GND
AC	Vdd	GND	SPEC	ED37	ED33	GND	ED28	GND	ED22	ED19	ED15	GND	ED12	ED08	ED05	GND	ED00	GND	D58	D54	S1	GND	Vdd

LANai 5

304-pin SuperBGA (top view)

CLOCKING

Pin	I/O	Description
CLK	I	**Clock:** The main clock input.
RCLK	I	**Reference Clock:** The additional clock-reference input.



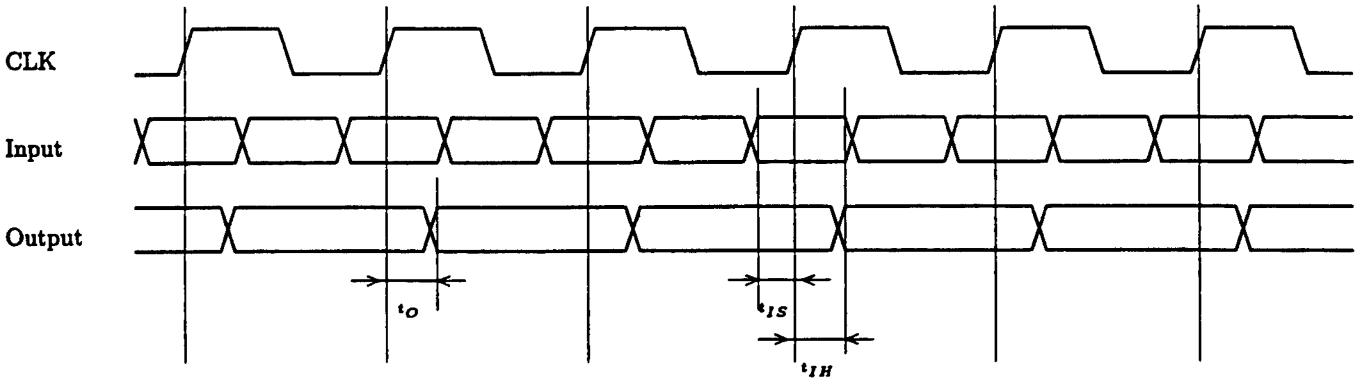
Symbol	Parameter	Min	Max
T_{CLK}	Clock period	20ns	20μs
T_{HCLK}	Clock half-period	10ns	10μs

CLK is the main clock input, and, except for the Myrinet-SAN interface (page 39), the entire chip works off of this clock. The internal chip clocks are derived from CLK, and there are several internal-clocking configurations, selected by the special register CLOCK (page 7):

<table>
<tr><th>Bit</th><th>Name</th><th>Description</th></tr>
<tr><td>7</td><td>PLL</td><td>When this bit is set, the on-chip PLL can be used to produce a symmetric on-chip clock, relaxing the minimum clock half-period requirement to 8ns. Minimum clock period requirement remains 20ns.</td></tr>
<tr><td>6-4</td><td>MULT</td><td>Frequency of the internal chip clock (ICLK) is a multiple of the frequency of the CLK signal. The three MULT bits determine the frequency-multiplication factor:
<table>
<tr><th>Value</th><th>Internal Clock</th></tr>
<tr><td>0</td><td>1 x CLK</td></tr>
<tr><td>1</td><td>1.5 x CLK</td></tr>
<tr><td>2</td><td>2 x CLK</td></tr>
<tr><td>4</td><td>3 x CLK</td></tr>
<tr><td>6</td><td>4 x CLK</td></tr>
</table></td></tr>
<tr><td>3-2</td><td>-</td><td>Reserved.</td></tr>
<tr><td>1</td><td>SELR</td><td>In systems where the CLK input is connected to a clock that may be stopped for arbitrarily long time periods (such as the PCI bus clock), the on-chip PLL requires an additional clock reference on the RCLK input. This clock reference must have duty cycle of 45-55%, maximum frequency of 60MHz, and its period must be smaller than the smallest possible period of CLK on that system.
The maximum clock period and half-period requirements remain 20μs and 10μs, respectively.</td></tr>
<tr><td>0</td><td>DBL</td><td>If this bit is 1, there are two LBUS memory accesses per clock cycle. If this bit is 0, there is one LBUS memory access per clock cycle.</td></tr>
</table>

Note: In the LANai5.0 chip, the PLL, SELR, and DBL bits are connected to the three input pins and cannot be modified by the programmer.

SYNCHRONOUS INPUTS AND OUTPUTS

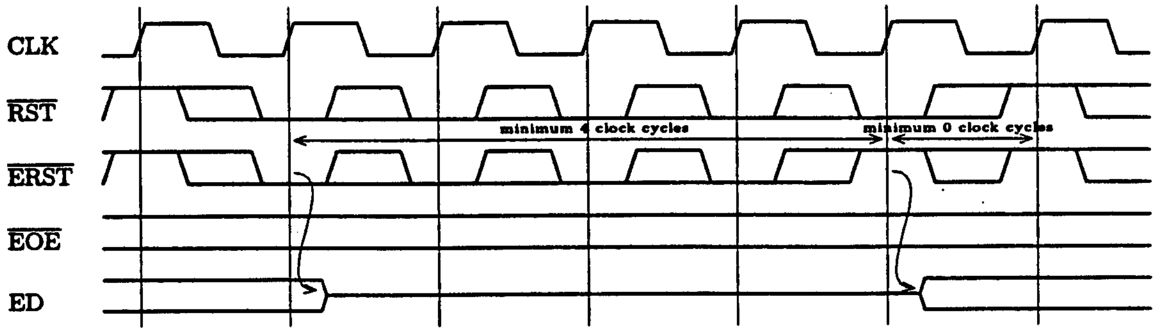


Symbol	Parameter	Min	Max
t_{IS}	Synchronous-input setup time	3ns	
t_{IH}	Synchronous-input hold time	0ns	
t_O	Synchronous-output delay (ICLK = CLK)	1ns	7ns
t_O	Synchronous-output delay (ICLK = 1.5 x CLK)	1ns	$\frac{2}{3}T_{CLK} + 7\text{ns}$
t_O	Synchronous-output delay (ICLK = 2 x CLK)	1ns	$\frac{1}{2}T_{CLK} + 7\text{ns}$
t_O	Synchronous-output delay (ICLK = 3 x CLK)	1ns	$\frac{2}{3}T_{CLK} + 7\text{ns}$
t_O	Synchronous-output delay (ICLK = 4 x CLK)	1ns	$\frac{3}{4}T_{CLK} + 7\text{ns}$

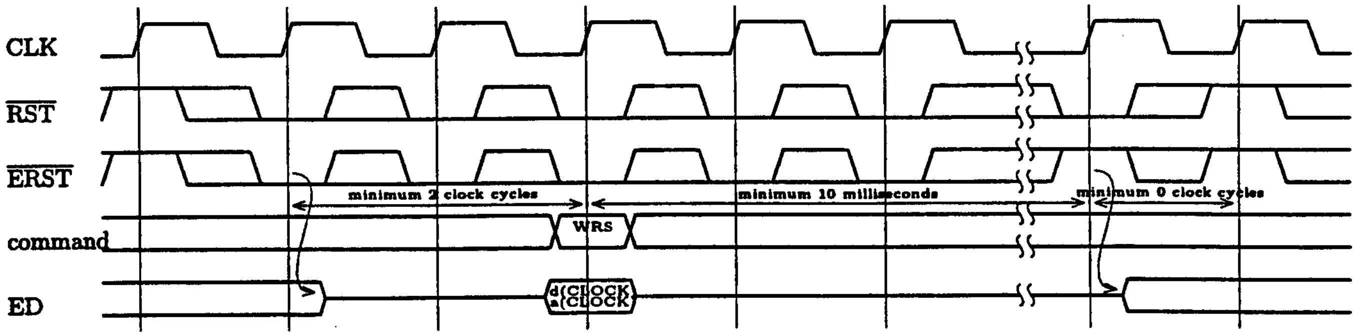
RESET SEQUENCE

Pin	I/O	Description
$\overline{\text{RST}}$	synch. I	**Reset:** The main reset input. During the power-on reset, the special register CLOCK, which controls the on-chip clock-generation, must be initialized. Note that, unlike with other special registers, both $\overline{\text{RST}}$ and $\overline{\text{ERST}}$ must be asserted while the CLOCK is written. This register need not be initialized during a non-power-on reset.
$\overline{\text{ERST}}$	synch. I	**EBUS Reset:** This additional reset input resets the LANai circuitry associated with EBUS access. When the $\overline{\text{RST}}$ input is asserted and $\overline{\text{ERST}}$ is not, the EBUS access is enabled so that, for example, code for the LANai processor can be loaded into the LBUS memory.

Non-Power-On Reset Sequence

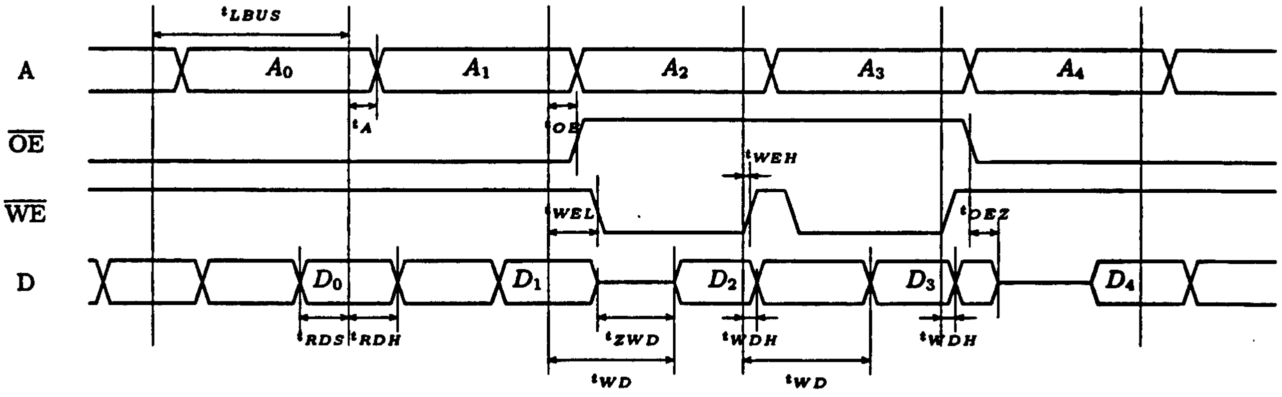


Power-On Reset Sequence



LBUS INTERFACE

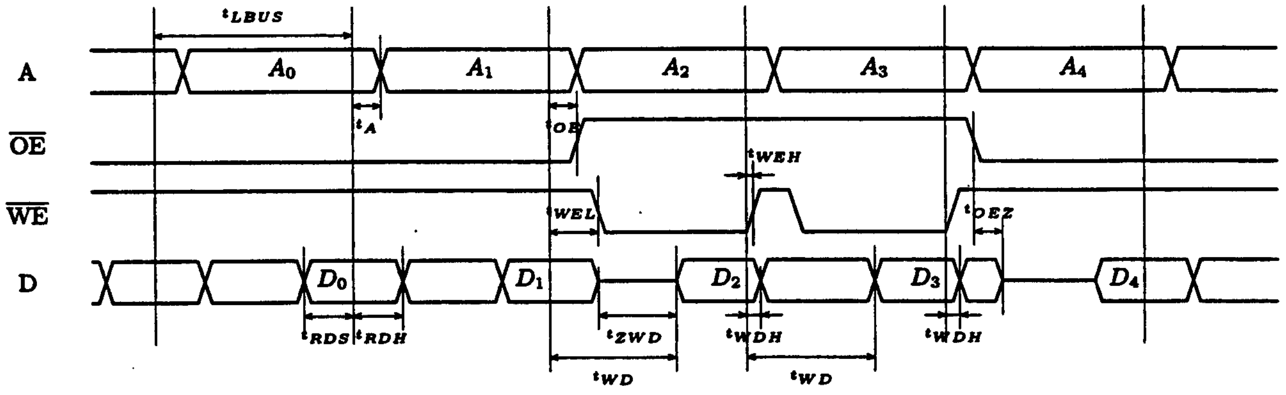
Pin	I/O	Description
D00 - D63	I/O	**LBUS Data:** 64-bit bi-directional data bus (bit 63 is the most significant).
A03 - A23	O	**LBUS Address:** The LBUS address pins. The internals of the LANai 5 chip support 32-bit addresses, but pin count limits the LBUS address space to 16 megabytes.
$\overline{A18}$, $\overline{A20}$, $\overline{A22}$	O	**LBUS Address Complement:** These three signals are provided for address decoding when using multiple banks of SRAM chips.
$\overline{WE0}$ - $\overline{WE7}$	O	**LBUS Write Enable:** 8 write enable signals, one for each byte in the 64-bit word. The LANai is a big-endian machine, and $\overline{WE0}$ corresponds to the smallest byte address, i.e., to the most-significant byte.
$\overline{OE}$	O	**LBUS Output Enable:** This signal controls the direction of the LBUS data bus.



Symbol	Parameter	Min	Max
T_{ICLK}	Period of the internal LANai clock (page 23).		
t_{LBUS}	LBUS memory cycle (DBL=0)	T_{ICLK}	
t_{LBUS}	LBUS memory cycle (DBL=1, PLL=1)	$T_{ICLK}/2 - 0.5$ns	
t_A	Address delay	out_L	$out_L + 0.5$ns
t_{OE}	Output-enable delay	out_L	$out_L + 0.5$ns
t_{WEL}	Write-enable assertion delay	$T_{WCLK} + out_L$	$T_{WCLK} + out_L + 0.5$ns
t_{WEH}	Write-enable deassertion delay	out_L	$out_L + 0.5$ns
t_{RDS}	Read-data setup time	1ns	
t_{RDH}	Read-data hold time	0ns	
t_{WD}	Write-data delay (LANai limited)	out	$out + 4$ns
t_{ZWD}	Write-data delay (SRAM limited)		out
t_{WDH}	Write-data hold time	0ns	0.5ns
t_{OEZ}	Data-drivers turn-off time	0ns	0.5ns
T_{WCLK}	CLOCK-register-controlled delay which ensures that the $\overline{WE}$ is asserted only after the address is stable	0ns	4ns

LBUS INTERFACE

Pin	I/O	Description
D00 - D63	I/O	**LBUS Data:** 64-bit bi-directional data bus (bit 63 is the most significant).
A03 - A23	O	**LBUS Address:** The LBUS address pins. The internals of the LANai 5 chip support 32-bit addresses, but pin count limits the LBUS address space to 16 megabytes.
$\overline{\text{A18}}$, $\overline{\text{A20}}$, $\overline{\text{A22}}$	O	**LBUS Address Complement:** These three signals are provided for address decoding when using multiple banks of SRAM chips.
$\overline{\text{WE0}}$ - $\overline{\text{WE7}}$	O	**LBUS Write Enable:** 8 write enable signals, one for each byte in the 64-bit word. The LANai is a big-endian machine, and $\overline{\text{WE0}}$ corresponds to the smallest byte address, i.e., to the most-significant byte.
$\overline{\text{OE}}$	O	**LBUS Output Enable:** This signal controls the direction of the LBUS data bus.



Symbol	Parameter	Min	Max
T_{ICLK}	Period of the internal LANai clock (page 23).		
t_{LBUS}	LBUS memory cycle (DBL=0)	T_{ICLK}	
t_{LBUS}	LBUS memory cycle (DBL=1, PLL=1)	$T_{ICLK}/2 - 0.5$ns	
t_A	Address delay	out_L	$out_L + 0.5$ns
t_{OE}	Output-enable delay	out_L	$out_L + 0.5$ns
t_{WEL}	Write-enable assertion delay	$T_{WCLK} + out_L$	$T_{WCLK} + out_L + 0.5$ns
t_{WEH}	Write-enable·deassertion delay	out_L	$out_L + 0.5$ns
t_{RDS}	Read-data setup time	1ns	
t_{RDH}	Read-data hold time	0ns	
t_{WD}	Write-data delay (LANai limited)	out	$out + 4$ns
t_{ZWD}	Write-data delay (SRAM limited)		out
t_{WDH}	Write-data hold time	0ns	0.5ns
t_{OEZ}	Data-drivers turn-off time	0ns	0.5ns
T_{WCLK}	CLOCK-register-controlled delay which ensures that the $\overline{\text{WE}}$ is asserted only after the address is stable	0ns	4ns

The LANai 5 chip, starting with version 5.2, includes the circuitry that enables it to delay the timing of the LBUS data bus relative to the LBUS address bus. Asynchronous SRAM chips typically guarantee that the read data will not change for a specified period of time after an address change. The LANai 5 chip can take advantage of this feature, which may relax the SRAM access-time requirement. The table below specifies the timing information in the general case, with the LBUS-data timing delayed from the LBUS-address timing by T_{LCLK}.

Symbol	Parameter	Min	Max
t_{LBUS}	LBUS memory cycle (DBL=0)	T_{ICLK}	
t_{LBUS}	LBUS memory cycle (DBL=1, PLL=1)	$T_{ICLK}/2 - 0.5$ns	
t_A	Address delay	out_L	$out_L + 0.5$ns
t_{OE}	Output-enable delay	$T_{LCLK} + out_L$	$T_{LCLK} + out_L + 0.5$ns
t_{WEL}	Write-enable assertion delay	$T_{WCLK} + out_L$	$T_{WCLK} + out_L + 0.5$ns
t_{WEH}	Write-enable deassertion delay	out_L	$out_L + 0.5$ns
t_{RDS}	Read-data setup time	$-T_{LCLK} + 1$ns	
t_{RDH}	Read-data hold time	T_{LCLK}	
t_{WD}	Write-data delay (LANai limited)	$T_{LCLK} + out$	$\max(T_{LCLK}, 4ns) + out$
t_{ZWD}	Write-data delay (SRAM limited)		out
t_{WDH}	Write-data hold time	T_{LCLK}	$T_{LCLK} + 0.5$ns
t_{OEZ}	Data-drivers turn-off time	0ns	0.5ns
T_{WCLK}	CLOCK-register-controlled delay which ensures that the $\overline{\text{WE}}$ is asserted only after the address is stable	0ns	4ns
T_{LCLK}	CLOCK-register-controlled delay which delays the LBUS data timing relative to the LBUS address timing	0ns	4ns

The following bits of the special register CLOCK (page 7) control T_{WCLK} and T_{LCLK}:

Bit	Name	Description
15-12	WCLK	These four bits control the timing of the $\overline{\text{WE}}$ pins, to ensure that they are asserted only after the address is stable.
11-8	LCLK	These four bits control the timing of the $\overline{\text{OE}}$ and D, to delay the LBUS data timing relative to the LBUS address timing.

Note: The WCLK timing can be arbitrarily adjusted to conform to the SRAM at hand. However, changing the LCLK timing may require that some additional, internal chip timing be adjusted as well. The information required is beyond the scope of this document.

EBUS INTERFACE

The LANai 5 EBUS is a 64-bit wide, synchronous, pipelined interface, optimized for burst memory access.

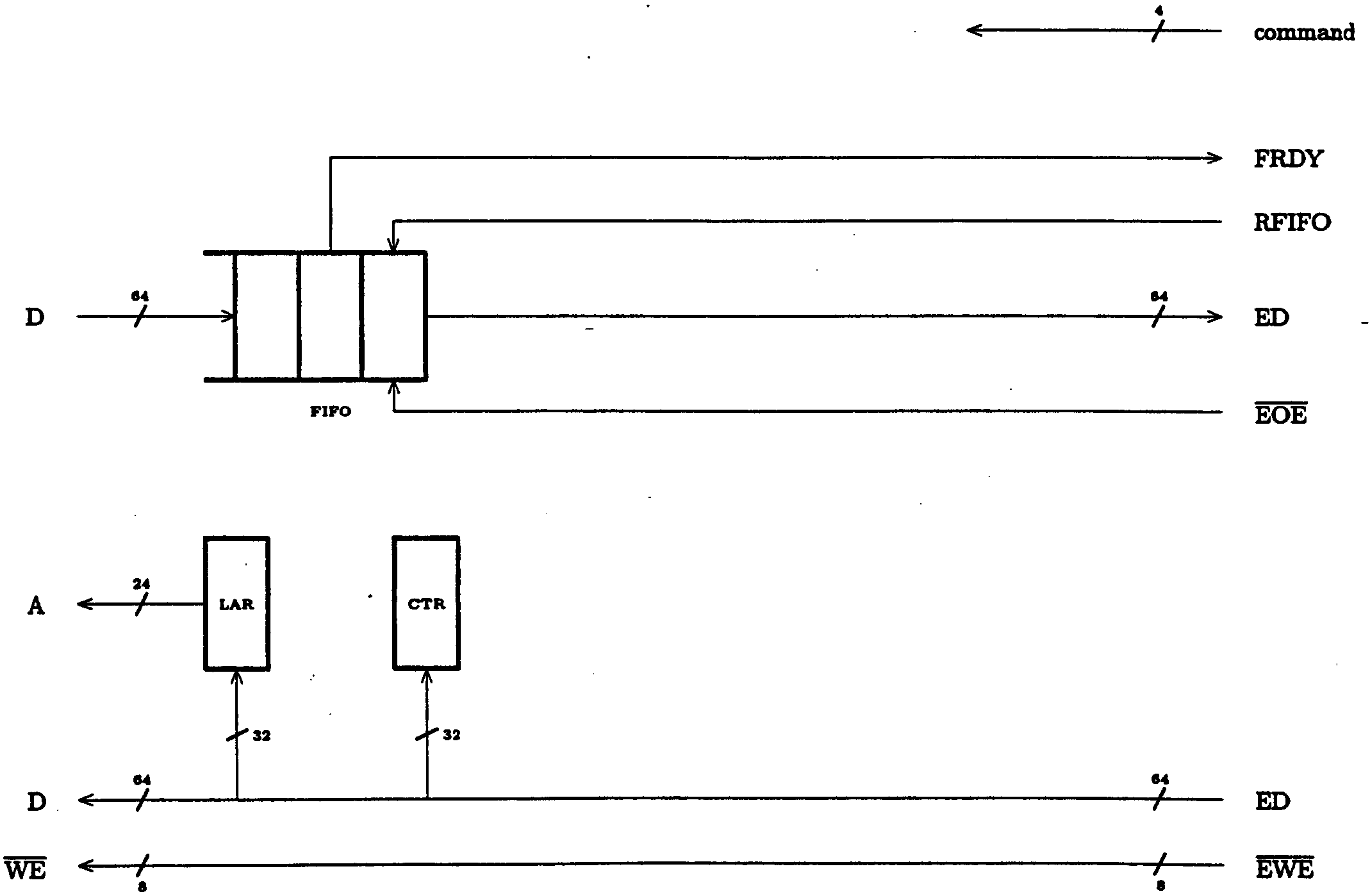
The simplest EBUS operation is writing a value into a LANai special register. Since the LANai special registers have at most 32 bits, and the LANai address space is also 32 bits, the address and data are simultaneously presented on the 64-bit ED bus.

Writing into the LANai LBUS memory is a two-stage operation: first the address of the memory access is written into the special register LAR (LBUS Address Register), and later the 64 data bits are presented on the ED bus, along with the request that they be written into the memory at the address specified by the LAR. The side effect of the memory-write request is that the LAR is incremented by 8, so that consecutive memory locations can be written without having to write the LAR again.

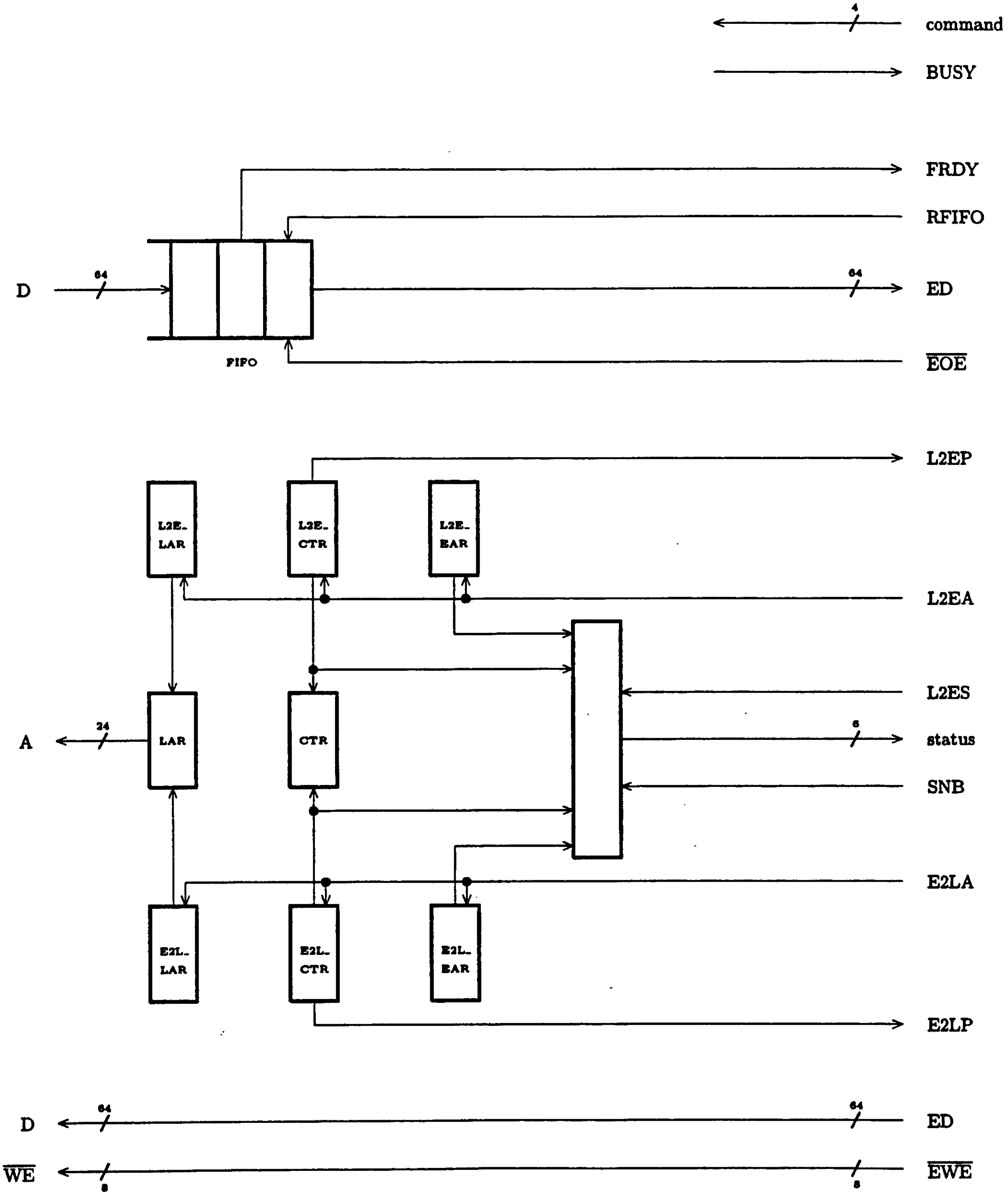
The read accesses of special registers and LBUS memory are done in a similar fashion, but the read data becomes available with the latency of three clock cycles. To be able to sustain the full EBUS bandwidth, the read data is placed into the three-double-words-deep FIFO. The most distinguishing characteristic of the EBUS interface is that the consumption of data from the FIFO is independent of the read requests, as long as the FIFO is kept from overflowing. For one particular, often-used case of reading a block of data from the LBUS memory, the LANai provides a mechanism that can be used to keep the FIFO from overflowing (page 32). The LBUS Access Counter (CTR) is used to specify the size of the data block.

The EBUS functionality described so far constitutes the core EBUS interface. There are several special registers (in addition to the LAR, CTR, and FIFO) that are used to provide additional functionality that may be useful when interfacing to standard I/O buses. However, all the additional features are experimental and may not be supported in future versions of the LANai chips.

CORE EBUS INTERFACE

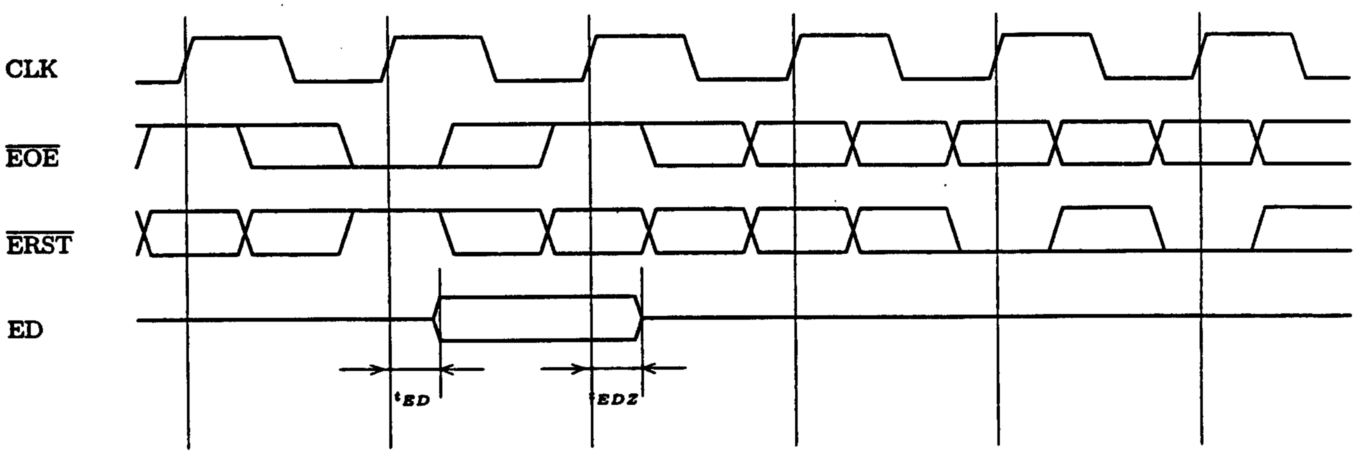


FULL EBUS INTERFACE



Pin	I/O	Description
$\overline{\mathrm{EOE}}$	synch. I	**EBUS Output Enable:** This signal controls the direction of the ED bus. If, at a rising CLK edge, the $\overline{\mathrm{EOE}}$ is asserted and the $\overline{\mathrm{ERST}}$ is not, the ED pins are outputs during the immediately following clock cycle.
ED00 - ED63	synch. I/O	**EBUS Data:** 64-bit bi-directional address and data bus (bit 63 is the most significant).
$\overline{\mathrm{EWE0}}$ - $\overline{\mathrm{EWE7}}$	synch. I	**EBUS Write Enable:** 8 write enable signals, one for each byte in the 64-bit word. The LANai is a big-endian machine, and $\overline{\mathrm{EWE0}}$ corresponds to the smallest byte address, i.e., to the most-significant byte. The values of these signals affect only the Write Memory EBUS command.

ED Timing

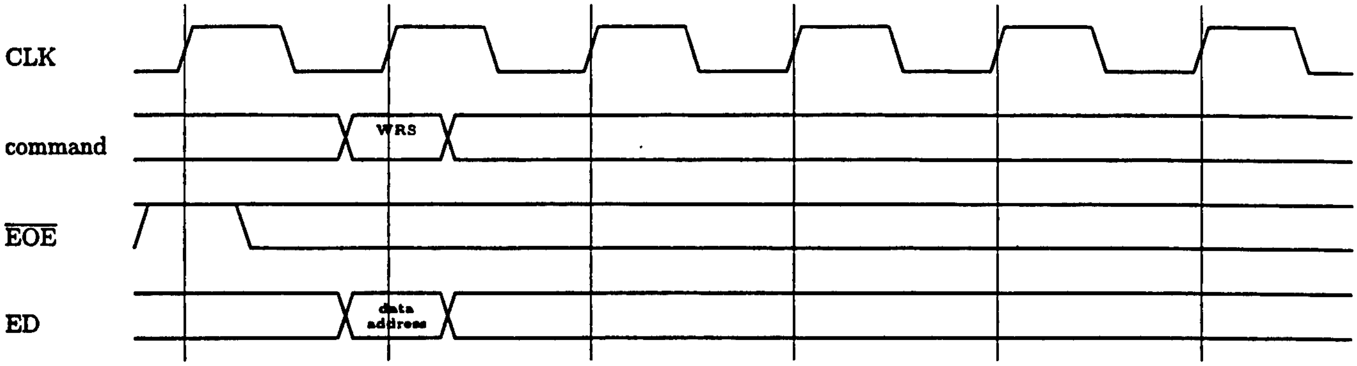


Symbol	Parameter	Min	Max
t_{EDZ}	Output release time	1ns	3ns
t_{ED}	Output delay	1ns	7ns

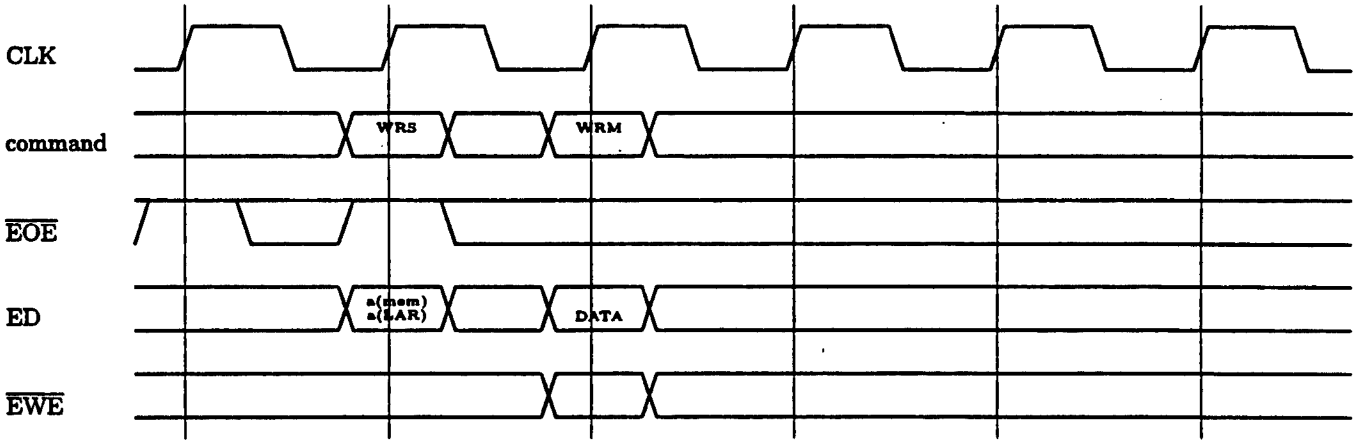
Pin	I/O	Description
WIN	O	**Chip Window:** This signal should be connected to a test point that can be used for timing observation of several internal chip signals. The special register DEBUG selects which internal signal is to be output on the WIN pin.
LED	synch. O	**LED Output:** This general-purpose output is controlled by the least-significant bit of the special register LED.
P0 P1 P2	synch. O	**PULSE Outputs:** This three outputs are controlled by the least-significant bits of the special register PULSE. When a value of 1 is written into such a bit, a one-clock-cycle-long pulse is generated on the corresponding output pin. There is no PULSE register in the LANai 5.0 version of the chip, and the three pins are used as inputs named DBL, SELR, and PLL, respectively (page 23).
INT	synch. O	**Interrupt Request:** This output is asserted if a bit in the special register ISR (Interrupt Status Register) and the corresponding bit in the special register EIMR (External Interrupt Mask Register) are both equal to 1.
WAKE	synch. I	**Wakeup:** When this input is asserted, the wake_int bit in the special register ISR (Interrupt Status Register) is set.
FRDY	synch. O	**FIFO Ready:** This signal indicates that the LANai EBUS FIFO has at least one word in it.
RFIFO	synch. I	**Read FIFO:** When both RFIFO and FRDY are asserted at a rising CLK edge, the LANai EBUS FIFO is advanced.
BUSY	synch. O	**EBUS Interface Busy:** This signal is asserted while the read DMA is going on. No EBUS command (except for NOP and STOP) may be issued while BUSY is asserted.
E2LP	synch. O	**EBUS→LBUS DMA Pending:** This signal is asserted when the LANai E2L_CTR special register has a non-zero value.
L2EP	synch. O	**LBUS→EBUS DMA Pending:** This signal is asserted when the LANai L2E_CTR special register has a non-zero value.
E2LA	synch. I	**EBUS→LBUS DMA Advance:** When this signal is asserted, E2L_LAR and E2L_EAR are incremented by 8, and E2L_CTR is decremented by 8. During a LANai-initiated E2L DMA, the E2LA signal must be asserted simultaneously with the WRM command, so as to keep the above three registers up to date.
L2EA	synch. I	**LBUS→EBUS DMA Advance:** When this signal is asserted, L2E_LAR and L2E_EAR are incremented by 8, and L2E_CTR is decremented by 8. For non-pipelined outside circuitry, this signal should be asserted simultaneously with the RFIFO signal, so as to keep the above three registers up to date. For applications where pipelined outside circuitry connects to an abortable system bus, when a system-bus transaction is aborted the outside circuitry may have already fetched one or more words of incoming read-DMA data from the LANai. The L2EA signal can be used such that the above three registers keep the correct record of the successfully completed transactions on the system bus. If so, upon an abort on the system bus, the outside circuitry may STOP any pending LANai DMA, flush the FIFO, and later resume the LANai DMA from the correct values of L2E_LAR, L2E_EAR, and L2E_CTR.
SPEC DMA WMEM RMEM	synch. I	**Special, DMA, Write Memory, Read Memory:** These 4 signals together specify the EBUS command, as specified below.

Command	SPEC	DMA	WMEM	RMEM	Description
NOP	0	0	0	0	Null Operation.
WRS	1	0	1	0	**Write Special Register:** The 10 least-significant bits of the ED bus specify the address of the LANai special register to write. The 32 most-significant bits of the ED bus specify the data to be stored into the register (special registers have at most 32 data bits).
WRM	0	0	1	0	**Write Memory:** The 64 ED bits specify the data to be stored into the memory location pointed to by the LANai LBUS Address Register (LAR). The $\overline{\text{EWE}}$ bits enable writes into their corresponding bytes within the 64-bit word. The LAR is incremented by 8, and the LBUS Access Counter (CTR) is decremented by 8.
RDS	1	0	0	1	**Read Special Register:** The 10 least-significant bits of the ED bus specify the address of the LANai special register to read. A 64-bit word, with its most-significant 32 bits equal to the value of the special register, and its least-significant 32 bit undefined, is written into the FIFO. If the FIFO is full, the value written into it is lost.
RDM	0	0	0	1	**Read Memory:** The 64-bit value of the memory location pointed to by the LAR is written into the FIFO. The LAR is incremented by 8 and the CTR is decremented by 8. If the FIFO is full, the value written into it is lost.
START	0	1	0	1	**Start Read DMA:** This command instructs the LANai to fetch the block of data starting at the memory location pointed to by the LAR and of size specified by CTR, and write it into the FIFO. The LAR is incremented by 8 and the CTR is decremented by 8 as each 64-bit word is read from the memory. The LANai guarantees that the FIFO is not overwritten. No EBUS command (except for NOP and STOP) may be issued while the read DMA is going on.
STOP	0	1	0	0	**Stop Read DMA:** This command instructs the LANai to stop any pending Read DMA and flush the FIFO.
E2L	1	1	1	0	**Start/Resume Write DMA:** This command copies the value of E2L_LAR into LAR and the value of E2L_CTR into CTR. This command is typically issued when the outside circuitry detects that the E2LP signal has been asserted. After issuing this command, the outside circuitry may provide the data to be written into the LBUS memory by issuing a sequence of WRM commands simultaneously with asserting the E2LA input.
L2E	1	1	0	1	**Start/Resume Read DMA:** This command copies the value of L2E_LAR into LAR and the value of L2E_CTR into CTR, and starts the Read DMA. This command is typically issued when the outside circuitry detects that the L2EP signal has been asserted. After issuing this command, the outside circuitry may fetch the data read from the LBUS memory by asserting the RFIFO signal. No EBUS command (except for NOP and STOP) may be issued while the read DMA is going on.

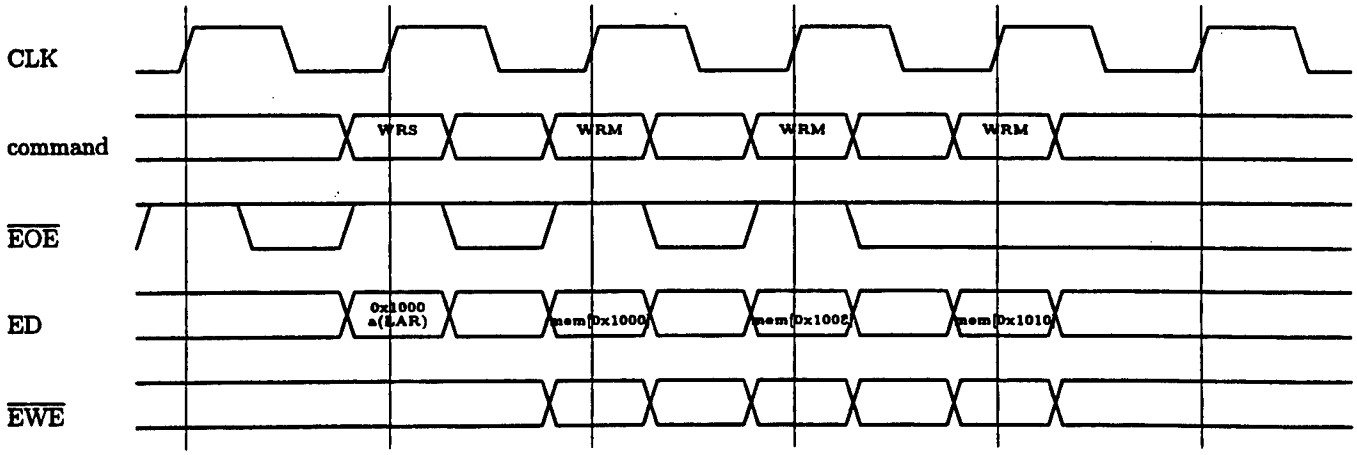
Write Special Register



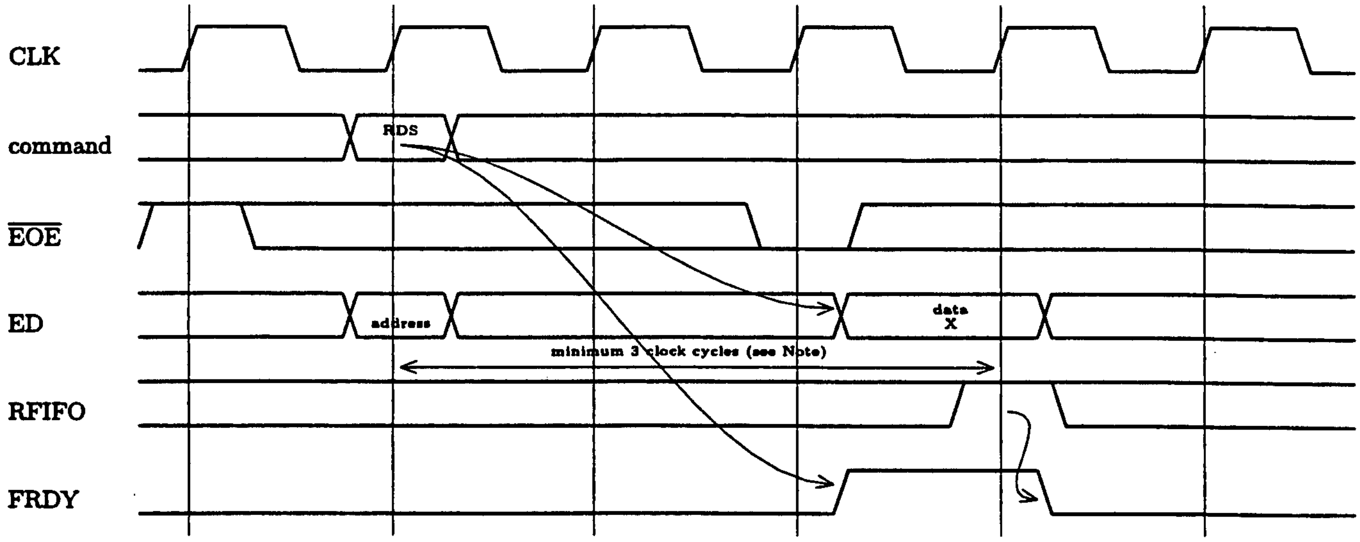
Write Single Word into the Memory



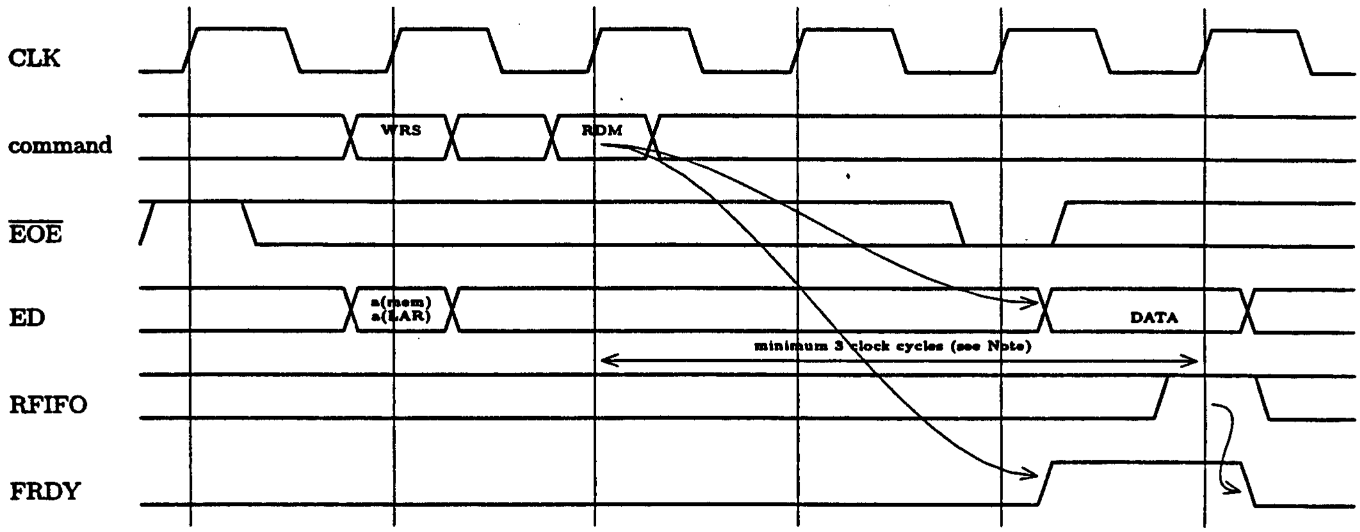
Write Successive Words into the Memory



Read Special Register

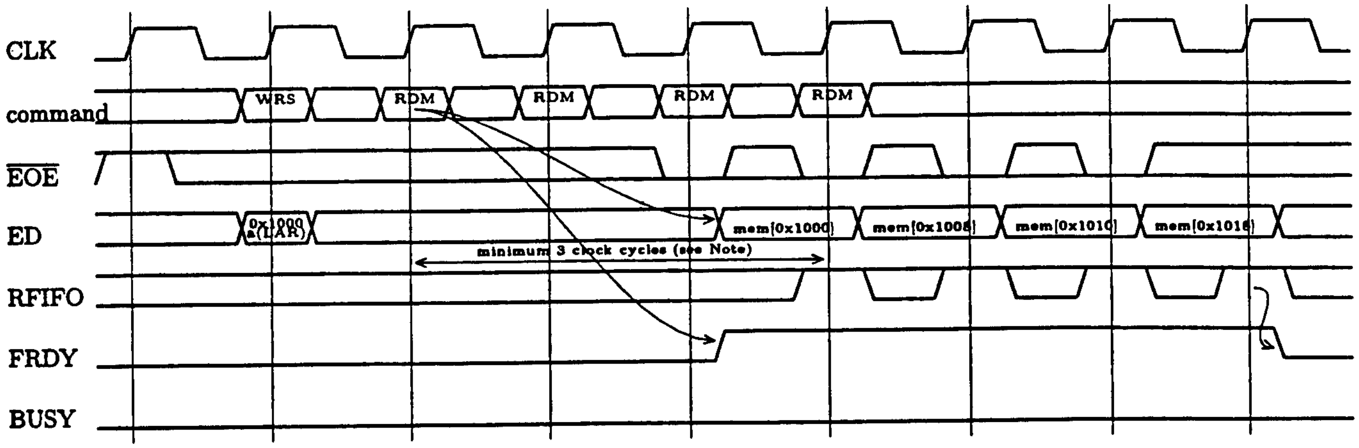


Read Single Word from the Memory

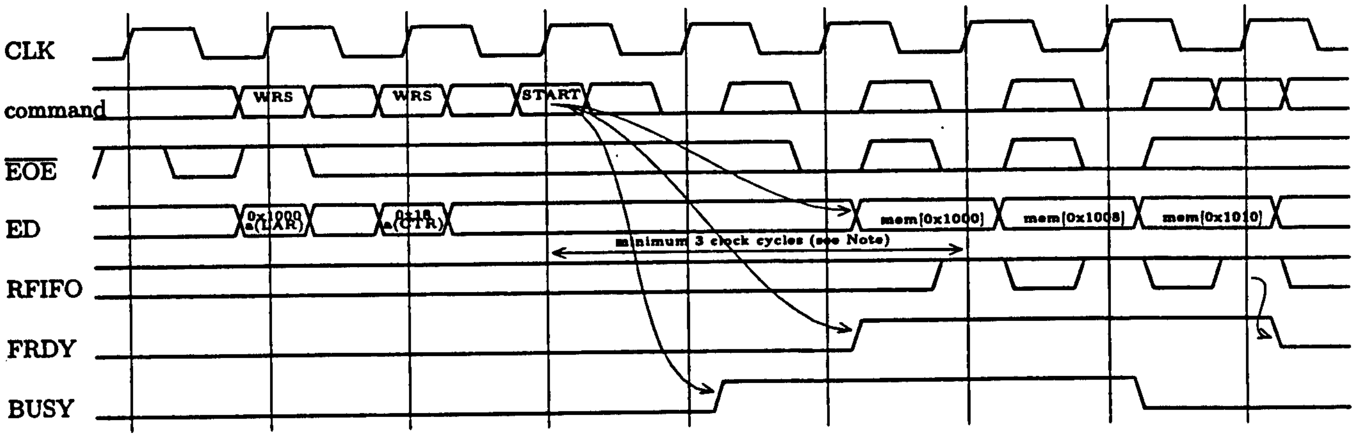


Note: When the internal-chip clock is 1.5 or 2 times the CLK, the read-access delay is equal to 2 CLK cycles. When the internal-chip clock is 3 or 4 times the CLK, the read-access delay is equal to 1 CLK cycle.

Read Successive Words from the Memory

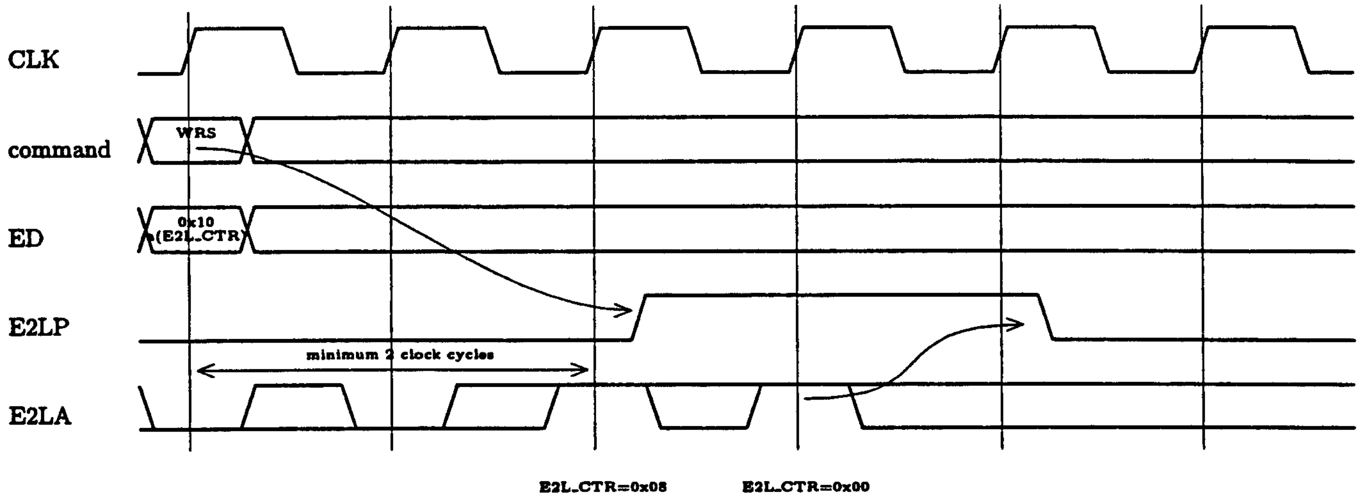


Read DMA

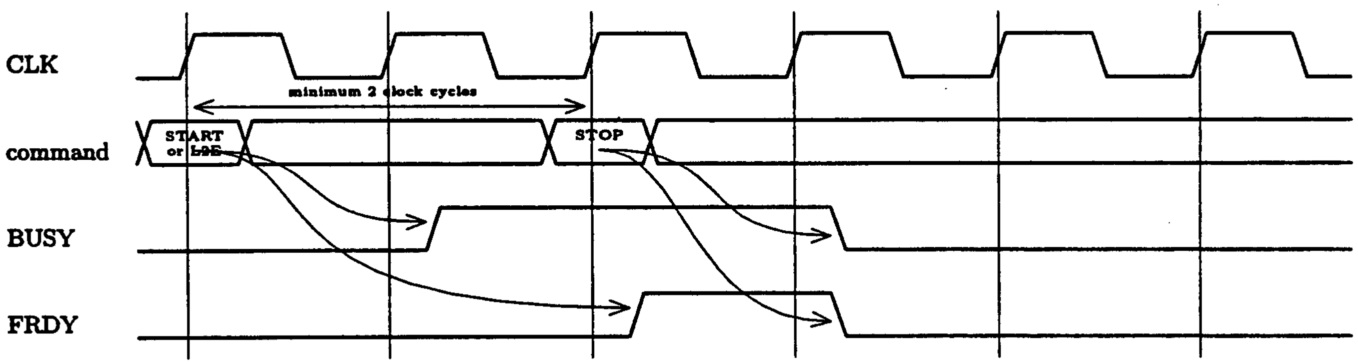


Note: When the internal-chip clock is 1.5 or 2 times the CLK, the read-access delay is equal to 2 CLK cycles. When the internal-chip clock is 3 or 4 times the CLK, the read-access delay is equal to 1 CLK cycle.

E2LP and E2LA Timing

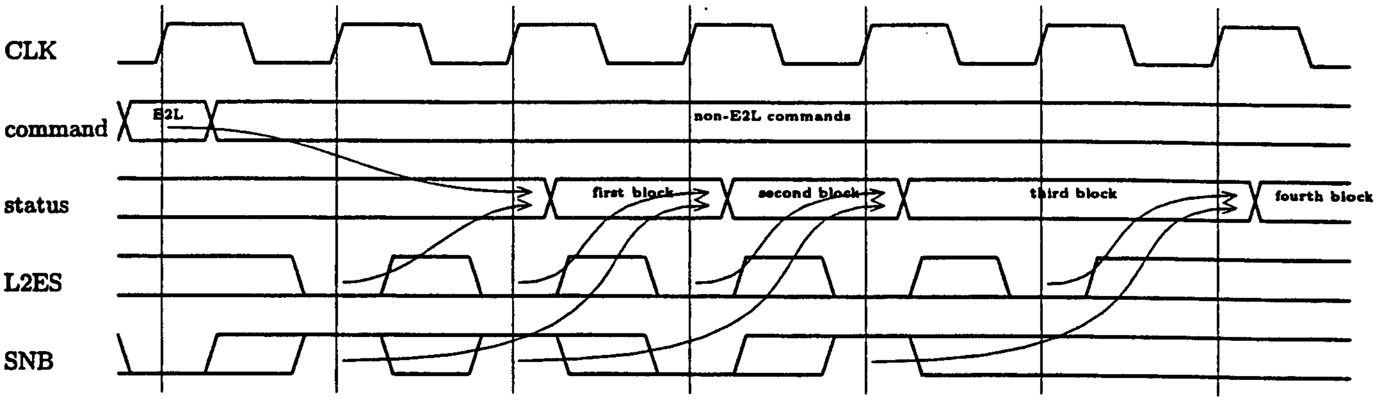


BUSY Timing



Pin	I/O	Description
L2ES	synch. I	**L2E Status Select:** This pin selects if the six status pins and the SNB pin described next correspond to the L2E DMA (L2ES=1), or to the E2L DMA (L2ES=0).
S16	synch. O	**Status-16:** A block of 16 double-words is pending.
S8	synch. O	**Status-8:** A block of 8 double-words is pending.
S4	synch. O	**Status-4:** A block of 4 double-words is pending.
S2	synch. O	**Status-2:** A block of 2 double-words is pending.
S1	synch. O	**Status-1:** A block of 1 double-words is pending.
SC	synch. O	**Status-C:** A block of cache-line size is pending.
SNB	synch. I	**Status of Next Block:** Some standard I/O buses (such as PCI and SBus) have burst-transfer modes that require that each data block be of size 2^N bytes, starting on 2^N-aligned address. Upon an L2E (if L2ES=1) or an E2L (if L2ES=0) command, the status pins reflect the size and alignment of the first available DMA block. After that, asserting the SNB pin requests that the status of the next available DMA block be output on the status pins. Please note that the LANai 5 provides no implicit synchronization between the block sizes output in response to SNB and the actual DMA transfers. This decoupling enables the outside circuitry to learn about the sizes of upcoming DMA blocks arbitrarily early. The PCI/Sbus selection and the cache-line size (for PCI), or the allowed burst modes (for Sbus), are written into the LANai special register BURST. Until this register is initialized, the default value of the BURST allows only 1-word transfers. The applications with Sbus-like interfaces use S16, S8, S4, S2, and S1 pins. Only one of these pins is asserted at any time and specifies the maximum aligned block that can be transferred next using burst mode. The applications with PCI-like interfaces use SC and S1 pins. Only one of these two pins is asserted at any time and specifies if the block is an aligned, cache-line-size block, or a single 64-bit word.

SNB Timing


CLK
command
E2L
non-E2L commands
status
first block
second block
third block
fourth block
L2ES
SNB

MYRINET SAN INTERFACE

	Pin	I/O	Description
Configuration	TCLK	I	**SAN Transmit Clock:** This clock input sets the data rate of the SAN output link. A byte is sent on every transition, so, for the nominal Myrinet SAN link rate of 160 megabytes per second, this is an 80 MHz clock. The allowed range for duty cycle is 45-55%. There is no restriction on the phase of TCLK with respect to any other clock (including TCLK on the other end of the SAN link). The frequency of TCLK must be within 1000ppm (0.1%) of 80MHz for compatibility with other Myrinet products.
	LVDD	power	**SAN Low-Voltage-Driver-Supply Voltage:** This pin should be connected to a 1.25V +/- 2% supply if it is to be compatible with other Myrinet SAN links. Power consumption varies with channel usage, 20mA minimum, 90mA maximum. If the SAN output drivers are shorted to GND, current draw can exceed 240mA.
	VTH	I	**SAN Input-Threshold Reference:** This reference input should be LVDD/2 ± 1%. Current draw is $1\mu A$ max.
	BIAS	I	**SAN Bias Reference:** For 3.3V operation, this pin should be fed 1.0mA. At that current level, the voltage on the pin will be approximately 1.4V. A 1.87KΩ resistor to 3.3V supply will achieve this.
	OAH	SAN O	**SAN Impedence Reference, High:** This pin should be connected to GND through a 50 ohm resistor.
	OAL	SAN O	**SAN Impedence Reference, Low:** This pin should be connected to LVDD through a 50-ohm resistor.
Input Channel	I0 - I7 ID	SAN I	**SAN Input:** The 8 data bits (I0-I7, I7 most significant) and the control bit (ID) are transition encoded. A transition on a data bit corresponds to the value of 1, no transition to the value of 0. A transition on the control bit corresponds to the data byte, no transition to the control symbol (see Myrinet SAN Link Specification). Each of these pins have a built-in 20KΩ pulldown resistor.
	OB	SAN O	**SAN Output Block:** This output is asserted to notify the connecting output SAN link channel that it must stop transmitting (see Myrinet SAN Link Specification).
Output Channel	O0 - O7 OD	SAN O	**SAN Output:** The 8 data bits (O0-O7, O7 most significant) and the control bit (OD) are transition encoded. A transition on a data bit corresponds to the value of 1, no transition to the value of 0. A transition on the control bit corresponds to the data byte, no transition to the control symbol (see Myrinet SAN Link Specification).
	IB	SAN I	**SAN Input Block:** When this input is asserted, the output channel stops transmitting (see Myrinet SAN Link Specification). This pin has a built-in 20KΩ pulldown resistor.

Myrinet SAN over VME P0

Chuck Seitz

presented at a DARPA meeting in Santa Fe, NM, March 1997 - work done earlier under AO # B861, Contract F30602-94-C-0270

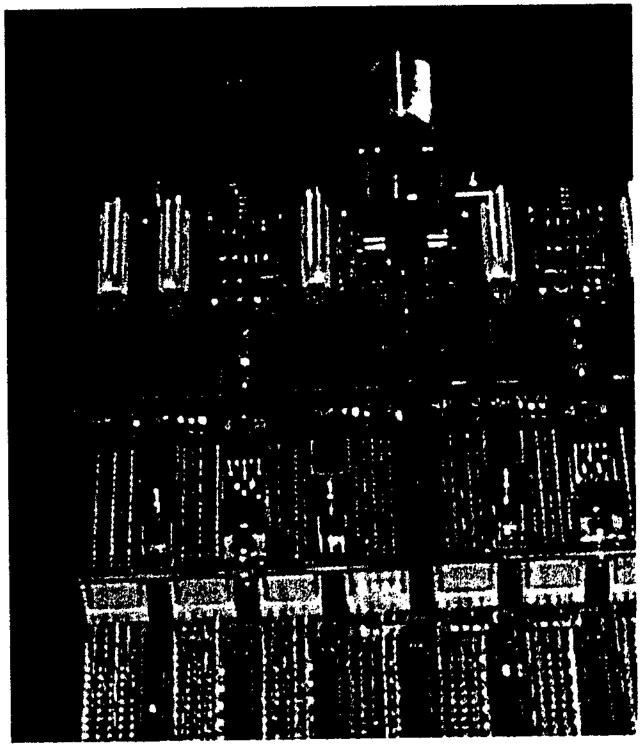
Myricom, Inc. 325 N. Santa Anita Ave. Arcadia CA 91006
818-821-5555 Fax:818-821-5316 http://www.myri.com

The Need...

- Very high, scalable, bisection data rates (many GB/s) between VME processor boards within and between subracks
- Myrinet-SAN on the VME front panel is a solution today, but can we provide an alternative to front-panel cables?
- The new opportunity: VME backplanes with the new P0 connector

A Solution – P0 Overlay Switch

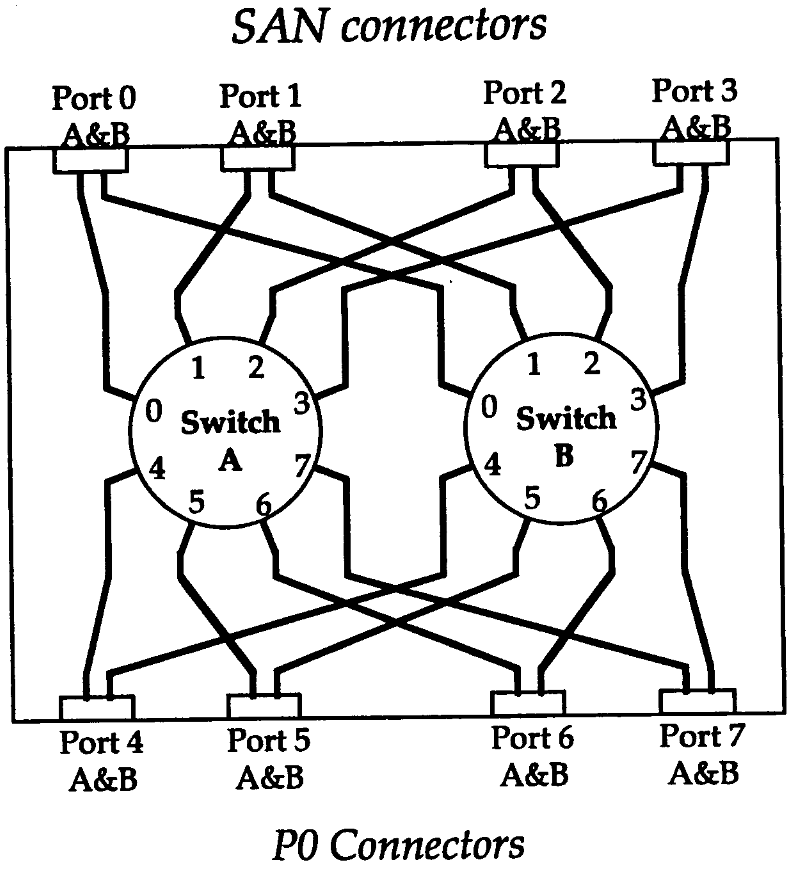
- ■ Developed jointly by Myricom and CSPI
- ■ Proposed as an ANSI standard under VITA auspices
- ■ COTS – available from Myricom and CSPI



Myricom

Myricom, Inc. 325 N. Santa Anita Ave. Arcadia CA 91006
818-821-5555 Fax:818-821-5316 http://www.myri.com

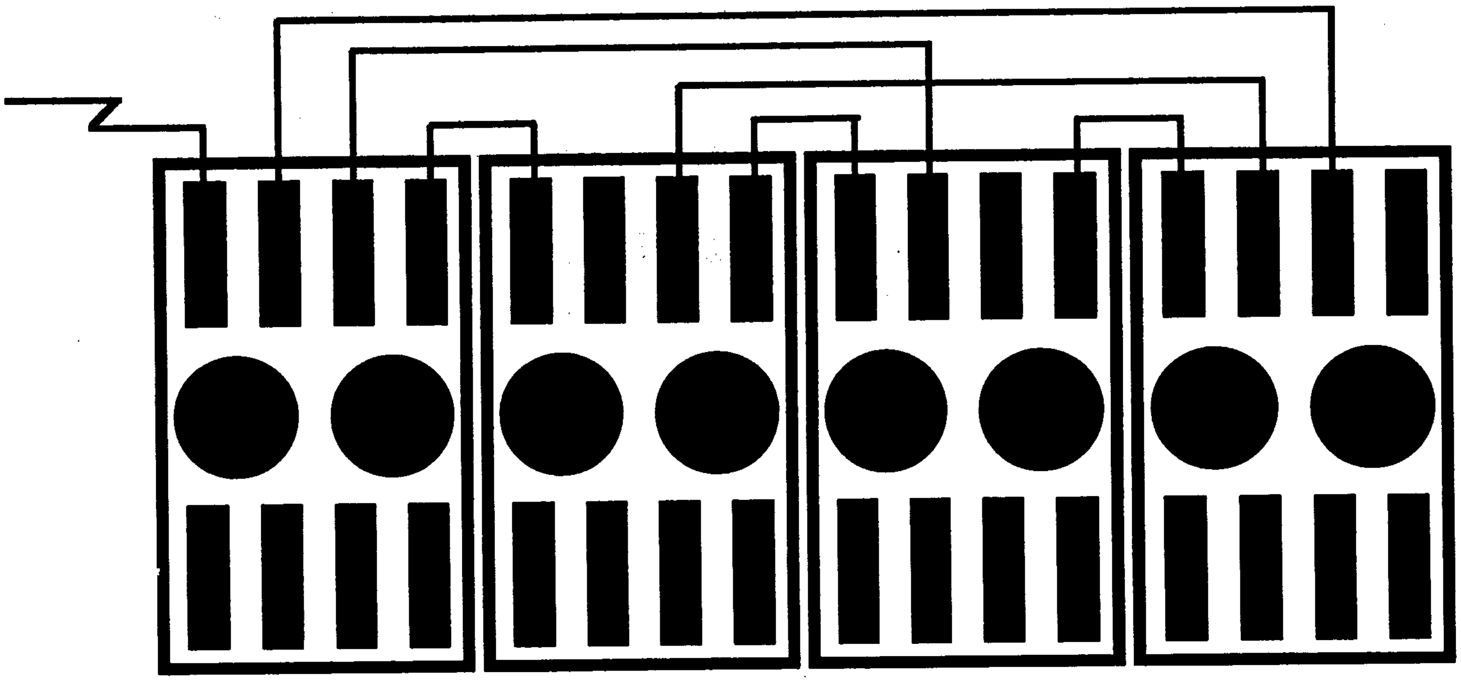
4-Slot Switch – block diagram



Performance

- Two links (160+160 + 160+160 MB/s) through P0
- Networks with high bisection data rates can be assembled with cables between P0-overlay switches (see next slide).
- Myricom has already characterized the P0 connector stackup and SAN cables for 320+320 MB/s links

Typical 16-slot configuration

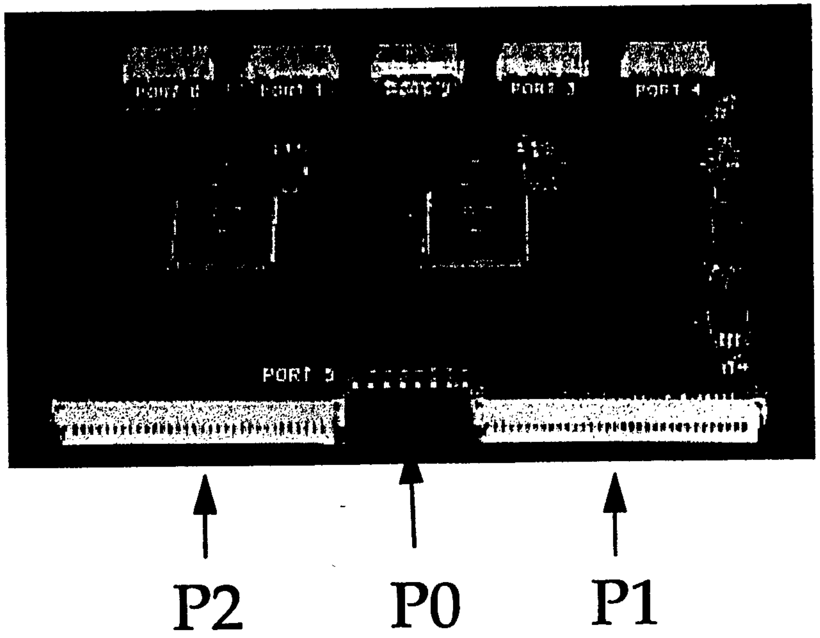


Myricom, Inc. 325 N. Santa Anita Ave. Arcadia CA 91006
818-821-5555 Fax:818-821-5316 http://www.myri.com

Backward Compatibility

- The active-backplane, P0-overlay switch can be used together with Myrinet-SAN (PMC boards) on the front panel, and may coexist with Raceway and other P2 buses

This M2M-VME-SW12 switch carries packet communication between the front panel and backplane-overlay switch.



Myricom

Myricom, Inc. 325 N. Santa Anita Ave. Arcadia CA 91006
818-821-5555 Fax:818-821-5316 http://www.myri.com

Useful to know or remember...

- ■ P0 backplanes are now available
  - ➤ for future systems
  - ➤ to upgrade present systems
- ■ Myricom will develop...
  - ➤ Bulkhead SAN-LAN converters
  - ➤ A variety of P0-overlay switches
- ■ Remember that VME is the FORTRAN of electronic packaging

Myricom

Scalable, Network-Connected, Reconfigurable, Hardware Accelerators for an Automatic-Target-Recognition Application

By

Wen-King Su Ruth Sivilotti
Young Cho Danny Cohen

With guidance and help from Sandia's

Brian Bray and Doug Doerfler

May 1998

The research described in this report was sponsored by the Defense Advanced Research Projects Agency. The earlier phases of the research, concentrating on the two-level-multicomputer architecture and hardware implementations, was sponsored under DARPA Order B861, and monitored by the USAF Rome Laboratory under contract number F30602-94-C-0270. The later phases of the research, concentrating on this ATR "challenge application," was sponsored under DARPA Order E258, and monitored by the US Army, Fort Huachuca, under contract number DABT63-96-C-0052.

Please address questions and comments about this report to <Cohen@myri.com>.

Myricom, Inc., 325 North Santa Anita Avenue, Arcadia, CA 91006 Tel: 626-821-5555
"http://www.myri.com" Fax: 626-821-5316

Scalable, Network-Connected, Reconfigurable, Hardware Accelerators for an Automatic-Target-Recognition Application

Introduction

Image processing, specifically Automatic Target Recognition (ATR) in Synthetic Aperture Radar (SAR) imagery, is an application area that requires tremendous processing throughput. In this application, data comes from high-bandwidth sensors, and the processing is time-critical. There is limited space and power for processing the data in the sensor platforms or in battlefield ground stations. DoD's strong push for using commercial-off-the-shelf (COTS) technology, the very high non-recurring engineering (NRE) costs for low volume ASICs, and evolving algorithms limit the feasibility of using custom special purpose hardware. In addition, a scalable system is required as the different sensor platforms have different image pixel rates and different mission requirements have different target recognition throughput needs per pixel.

To meet this challenge, Myricom, under DARPA funding, has developed a compact scalable system for high-performance implementation of the SAR ATR algorithms that were developed by Sandia National Laboratories (SNL). This system is based on the use of multiple, concurrent, specially programmed, reconfigurable computing nodes, interconnected by Myrinet. To achieve high efficiency, through the exploitation of the unique characteristics of this algorithm (e.g., operations on single bits), special FPGA-based computing nodes were developed by Myricom and delivered to SNL.

In this report, we describe the ATR algorithm implementation using FPGA accelerators. We describe the ATR algorithm that was implemented, the implementation on a single FPGA, how the FPGA nodes are connected to make a scalable system, and report both simulated and measured performance.

This system is a two-level computing system whose first level provides the general purpose infrastructure of network interfacing, message handling, mapping, initialization, etc., by the LANai microprocessor. Its second level provides the application-specific computing, which in this case is performed by the Lucent ORCA FPGA.

The sections of this report describe:

1. Overview of the system
2. The computation task, as developed by Sandia National Laboratories
3. Simulations
4. Adapting the algorithm to the structure of the FPGA computing nodes
5. Data flow through the system
6. Software functionality
7. Hardware
8. Performance, present and future
9. Summary
10. Conclusions

1. Overview of the System

The general flow of information in the entire ATR system is:

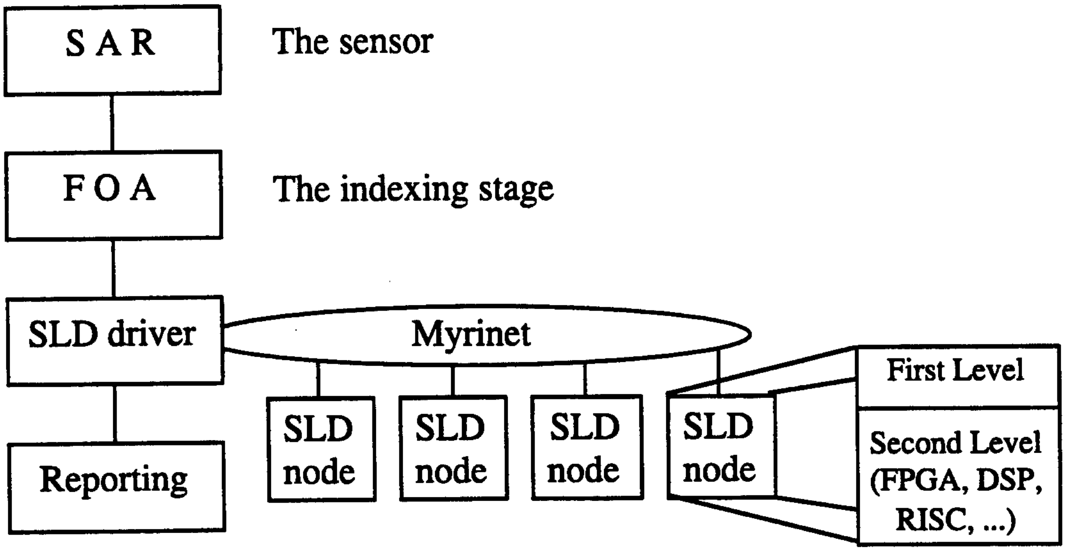


Figure 1: The overall ATR system

The FOA subsystem (for "Focus of Attention") handles the image as received by the SAR system (after its initial digital signal processing) and identifies locations where targets might be.

The SLD subsystem further checks these suspicious locations against target templates, by performing the SLD ("Second-Level Detection") operation. The SLD operation is "embarrassingly parallel", and could be performed by a multitude of various concurrent processing nodes which could be general purpose processors (such as Single-Board Computers, SBCs), or special purpose units.

The SLD driver then reports its best matches to the reporting section of the ATR system, which decides what to report according to various considerations.

The focus of the Myricom work is the SLD operation, which is checking ("correlating") image chips against target templates.

Sandia's current ATR systems are based on heterogeneous two-level multicomputers, microprocessors and DSP chips that are linked by Myrinet in a system area network (SAN) configuration providing a high level of fault tolerance, scalability, and load sharing. Myrinet is a scalable high-bandwidth network based on highly capable network interfaces and non-blocking crossbar switches. The first level of computing is the general network interface, and the second level of computing is the ATR specific programs running on microprocessors (such as a 200MHz Motorola 603ev PowerPC) and DSPs (such as a SHARC).

Myricom's implementation uses FPGAs for computing at the second level.

We have developed high-performance, FPGA-based, compact, reconfigurable computing nodes to perform the SLD tasks.

We have also developed a VME-6U baseboard with an 8-port crossbar chip (Xbar8) that connects 4 external Myrinet links with 4 connectors for mezzanine ("daughter") boards. These boards are the FPGA nodes described in this report.

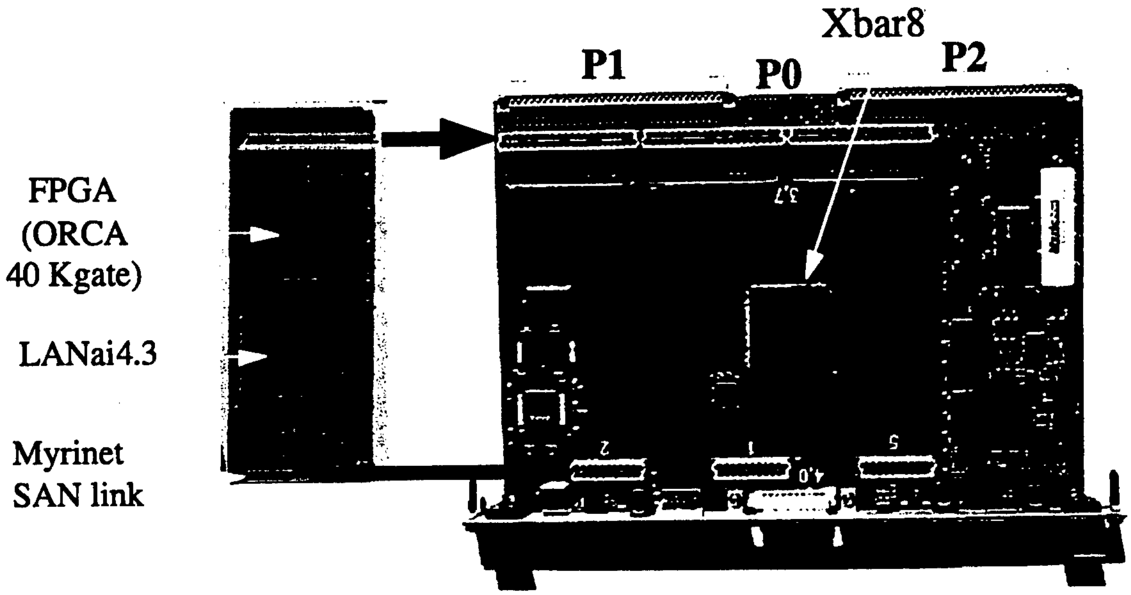


Figure 2: An FPGA mezzanine node (left) and a baseboard (right) with one node plugged to it

2. The Computation Task, as Developed by Sandia

Sandia National Laboratories' (SNL) real-time SAR ATR systems use a hierarchy of algorithms to reduce the processing demands for the SAR images, to yield a high probability of detection (Pd) and a low false alarm rate (FAR).

The first step in the SNL algorithm is a Focus of Attention (FOA) algorithm that runs over a down-sampled version of the entire image to find regions of interest that are of approximately the right size and brightness. The regions of interest are then extracted and processed by an indexing stage to further reduce the data stream which includes target hypotheses, orientation estimations, and target center locations. The surviving hypotheses have the full resolution data sent to an identification executive that schedules multiple identification algorithms and then fuses the results of the multiple identification algorithms.

The algorithm that we have implemented, using FPGA accelerators, is an indexing algorithm called Second-Level Detection (SLD). It is used for finding targets in-the-clear, not for camouflage, concealment or deception (CC&D) scenarios.

The SLD task is to take the extracted imagery (an image chip), match it against a list of provided target hypotheses, and return the hit information for each image chip consisting of the best two orientation matches, the degree of matching, the corresponding pixel location, and information about which target hypothesis gave rise to these two best matches.

SLD is a binary silhouette matcher that has a bright mask and a surround mask that are mutually exclusive. The bright mask and surround mask are 32x32 bitmaps, each having only about 100 non-zero pixels (about 10% of these bitmaps).

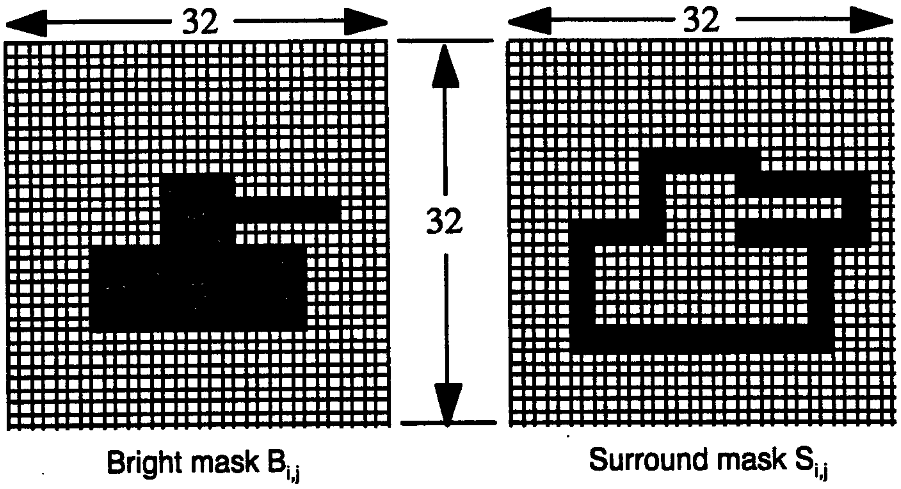


Figure 3: The two masks that are defined for every template

The system has a database of target models. For each target, and for each of a few elevations (typically 3) of the target, 72 templates are defined corresponding to all-around views of the target. The orientations of adjacent views are separated by 5 degrees.

Each template is composed of several parameters and two masks, a "bright mask" and a "surround mask", where the former defines the image pixels that should be bright for a match, and the latter defines the ones that should not. These masks are mutually exclusive. Being "bright" is defined relative to a dynamic threshold (to be described later).

The FOA stage identifies interesting image chips, and composes a list of targets suspected to be in that chip. Having access to range and altitude information, the FOA algorithm also determines the elevation for that chip, without having to identify the target first.

The FOA tasks the SLD stage to evaluate the likelihood that the suspected targets are actually in the given image chip, and exactly where. To do so, the FOA defines tasks for the SLD stage, where each task is composed of an image chip, a suspected target with its elevation, one or two orientation intervals, and a few parameters.

Upon receiving these tasks, the SLD unit matches all the stored templates for this target and elevation and the applicable orientations with the image chip, and computes the level of matching (the "hit-quality"). The two hits with the highest quality are reported to the SLD driver as the most likely candidates to include targets. For each hit the template ID (specifying the target type, its orientation, and its elevation), the exact position of the hit in the search area, and the hit quality are provided, too.

After receiving this information the SLD driver reports this information to the ATR system.

The size of the bitmaps of the target-orientation templates is 32x32.

The size of the image chips is 64x64 8-bit deep pixels. The FOA algorithms guarantee that the target (if any) is located in the image chip such that a 6 pixel margin around the chip is guaranteed not to include the target.

Hence, the area of interest in an image is 52x52 pixels.

In a 52x52 area there are 21x21 possible places to position a 32x32 mask area (52-32+1=21). This defines a search-area of 21 search-rows, each 21 position wide, as illustrated in the following figure.

The position (i,j) in the search area corresponds to positioning the lower left corner of the mask over the pixel (i,j) of the image.

Hence, 2x21x21=882 matches (of an image and a mask) have to be performed for each orientation that is specified in the matching task.

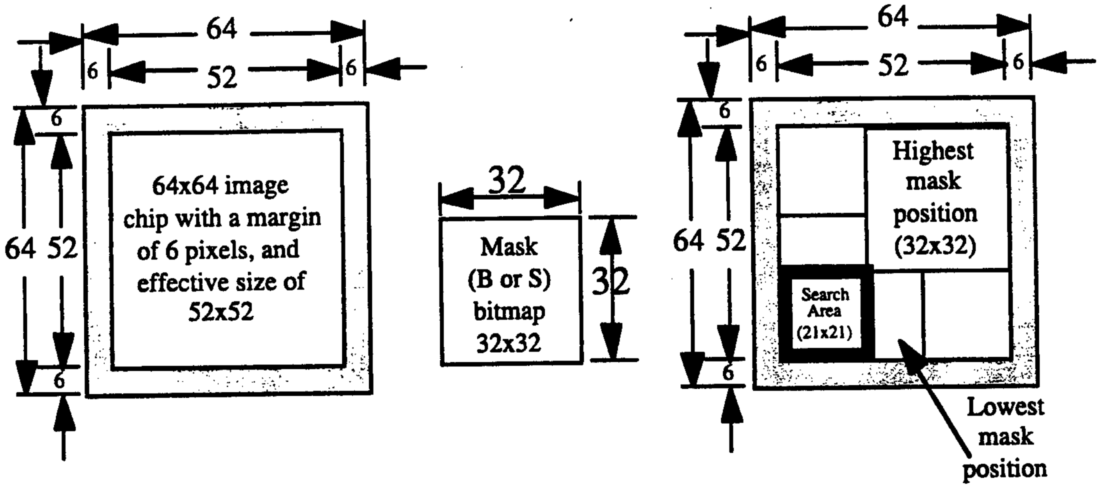

64
6
52
6
6
64
52
6
64x64 image chip with a margin of 6 pixels, and effective size of 52x52
32
Mask (B or S) bitmap 32x32
32
64
6
52
6
6
64
52
6
Highest mask position (32x32)
Search Area (21x21)
Lowest mask position

Figure 4: The relation between image chips, masks, and the search area

We use the following notation:

Image:	Image	M(a,b)	0...51 (or 6...57 inside of 0...63)
Templates:	Bright Mask	B(u,v)	0...31
	Bright Count	BC	
	Surround Mask	S(u,v)	0...31
	Surround Count	SC	
	Template bias	Bias	
	Minimal threshold	THmin	
	Maximal threshold	THmax	
	Minimal bright sum	BSmin	
	Minimal surround sum	SSmin	
Search Area:	Shape Sum	SM(i,j)	0...20
	Threshold	TH(i,j)	0...20
	Bright Sum	BS(i,j)	0...20
	Surround Sum	SS(i,j)	0...20
	Hit Quality	Q(i,j)	0...20
Performance:	(template/sec) per node	TSN	

The right column indicates the range for both variables.

The purpose of the first step in the SLD algorithm (called the "shape sum") is to distinguish the target from its surrounding background. It consists of adaptively estimating the illumination (energy under the bright mask) for each position in the search area assuming that the target is at that orientation and location. If the energy is too little or too much then no further processing for that position for that template match is required. Hence, for each mask position in the search area, a specific threshold value is computed (not one threshold value for the entire image nor for the entire search area).

The purpose of the next step in the SLD algorithm is to roughly distinguish the target from the background by thresholding each image pixel with respect to the threshold of the current mask position, as computed before. The same pixel may be above the threshold for some mask positions, but below it for others.

This threshold calculation is to determine what is really a bright pixel and what is a surround pixel. The calculation consists of dividing the shape sum by the number of pixels in the bright mask (i.e., finding the average brightness under the bright mask) and subtracting a template specific constant (Bias).

The pixels under the bright mask that are ***greater than or equal to*** the threshold are counted, and if this count exceeds the minimal bright sum (BSmin) the processing continues. Now the pixels under the surround mask that are ***less than*** the threshold are counted. If this count exceeds the minimal surround sum (SSmin) it is declared a hit. The quality of the hit is the average of the percent of bright and surround pixels that were correct.

The 5 computing tasks {Pk, k=1...5}, for each position (i,j) in the search area (i,j=0...20), are:

P1: The shape sum at (i,j) is: $SM(i,j)=\sum_{u=0}^{31}\sum_{v=0}^{31}B(u,v)M(i+u,j+v)$

P2: The threshold at (i,j) is: $TH(i,j)=\frac{SM(i,j)}{BC}-Bias$

where BC is the number of 1's in the bright mask, and $Bias$ is a template-specific value.

P3: The bright sum at (i,j): $BS(i,j)=\sum_{u=0}^{31}\sum_{v=0}^{31}B(u,v)[\ M(i+u,j+v)\geq TH(i,j)\ ]$

where [TRUE]=1 and [FALSE]=0.

P4: The surround sum at (i,j): $SS(i,j)=\sum_{u=0}^{31}\sum_{v=0}^{31}S(u,v)[\ M(i+u,j+v)< TH(i,j)\ ]$

A valid hit is at (i,j) if the 4 following conditions hold:

$$\begin{aligned}TH(i,j) &\leq TH\max\\ TH(i,j) &\geq TH\min\\ BS(i,j) &\geq BS\min\\ SS(i,j) &\geq SS\min\end{aligned}$$

P5: Finding and reporting the 2 valid hits with the highest hit quality.

The hit quality is defined by $Q(i,j)=\frac{1}{2}\left(\frac{BS(i,j)}{BC}+\frac{SS(i,j\ )}{SC}\right)$

where SC is the number of 1's in the surround mask.

3. Simulations

In order to verify our understanding of the SNL algorithm, we first implemented it in C, and ran it on a sample data set that we received from SNL. Our simulations reproduced the expected results received from SNL.

Over time this algorithm simulator has evolved into a full hardware simulator and verifier. It also allowed us to investigate various tradeoffs without actually implementing them in hardware.

This data set from SNL includes 2 targets, each with 72 templates, for 5-degree orientation intervals. Hence, in total we have 144 bright masks and 144 surround masks, each a 32x32 bitmap. The data set also includes 16 image chips (each 64x64 pixels at 1 byte/pixel).

Given a template and an image, there are 21x21=441 matrix correlations that must take place for each mask. This corresponds to 21 search rows, each 21 positions wide. The total number of search-rows which we correlate is:
(2*72 templates)*(16 images)*(21 (search-rows/image)/template)=48384 search-rows.

By analyzing the results of the simulations we have learned several important lessons:

Lesson-1: Check SS before BS

Using the given data the simulation yields:

Bad TH search-rows:	28841	(60%)
Bad SS search-rows:	19474	(40%)
Bad BS search-rows:	2	(0%)
Good search-rows:	67	(0%)
Total search-rows:	48384	(100%)

A bad-TH row is a row where for each j: (TH(i,j)>THmax) or (TH(i,j)<THmin).

A bad-SS row is a row that is not a bad-TH row, and where for all j: SS(i,j)<SSmin

A bad-BS row is a row that is neither a bad-TH row, nor a bad-SS row,
and where for all j: BS(i,j)<BSmin

Therefore it's better to check for bad-SS (i.e., if for all j: SS(i,j)<SSmin) before checking for bad-BS (i.e., if for all j: BS(i,j)<BSmin).

The low rejection rate by (BS(i,j)<BSmin) is the result of TH being computed using only the B-mask, regardless of the S-mask. TH(i,j) is computed exactly by the same pixels that are used for computing BS(i,j).

In this data set, only 67 search-rows passed all these 4 conditions (as entire rows), and out of their 67*21=1407 positions, only 84 passed them (as individual positions), and were declared as hits.

Lesson-2: Skip zero mask rows

Each masks has 32 rows. However, many masks have all-zero rows which can be skipped. By storing with each template a pointer to its first non-zero row we can skip directly to the first non-zero row "for free". Embedded all-zero rows are also skipped. But the computation to find them is hidden in the pipeline.

The simulation tools showed that, for the template set that we have received, this optimization significantly reduces the range for u (see figure below).

If the mask is B(u,v) then the range of u is its "length", and the range of v is its "width".

This skipping works best on "short" masks (i.e., masks with a small "length"), such as on the mask on the left in the following figure.

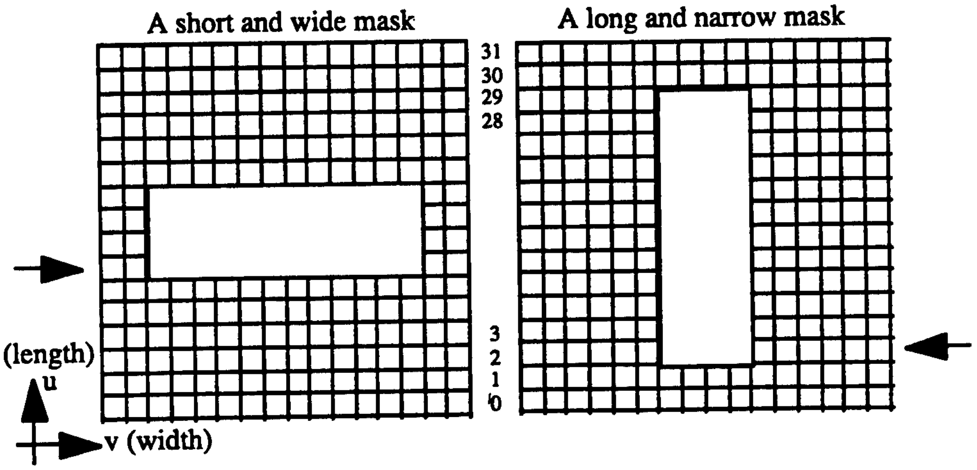


Figure 5: Short (left) and long (right) masks.

The data set has 72 template/target for 2 targets, for a total of 72*2=144 templates. Each template has 32 rows (of both masks) for a total of 144*32=4608 rows. Out of these 4608 bright-rows and 4608 surround-rows there are only 2206 non-zero bright-rows and 2815 non-zero surround-rows.

As expected, there are less non-zero bright-rows than surround-rows because the bright mask defines the body of the target, and the surround mask defines the external background around the target. Since the bright mask is used twice (for both TH and BS) whereas the surround mask only once (for SS) skipping the zero rows reduces the number of row operations from 4608+4608+4608=13824 to 2206+2206+2815=7227, which is only 52.3%, a saving of 1.9X.

The number of "internal" zero rows of the templates, according to our data, is very small, only 146 internal zero rows out of 9216 mask rows.

Lesson-3: : Skip zero mask columns

Just as Lesson-2 reduces the range for u (along the "length" of the masks) it is possible also to reduce the range of v (along the "width" of the masks) by skipping zero columns.

As seen later, our FPGA implementation works on an entire search-row concurrently. Hence, skipping rows reduces time, but skipping columns reduces the number of active elements that work in parallel, yielding no saving. Therefore, Lesson-2 is beneficial to our implementation, but Lesson-3 is not. We benefit from short masks (regardless of their width) but not from long masks.

Lesson-4: Transposing narrow masks

In order to benefit from narrow masks (as suggested in Lesson-3) it is possible to transpose narrow masks and correlate them against transposed images, allowing the same time savings for narrow masks as could be achieved for short masks (without rotation).

Hence, narrow masks (i.e., with a width that is smaller than their length) could be stored rotated, and correlated with rotated images, with the benefits of Lesson-2. If the number of templates that are to be matched with the image is large, the rotation task is justified.

4. Adapting the Algorithm to the Structure of the FPGA Computing Nodes

Each computing node has a LANai4 RISC processor, 512KB SRAM, and an FPGA (ORCA, 40K gate), as shown in the following figure.

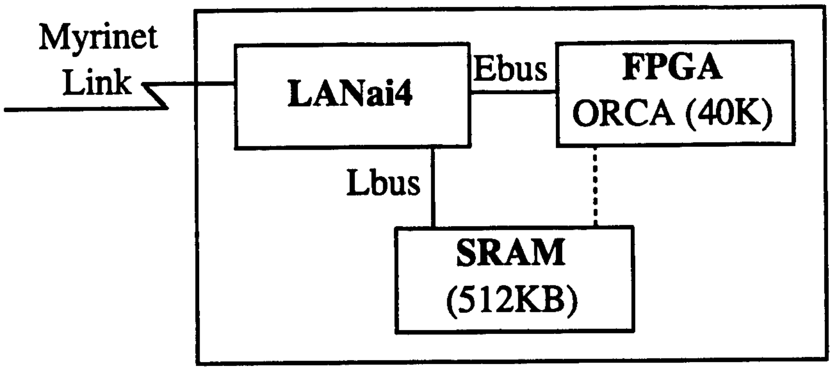


Figure 6: The functional structure of the FPGA node

When we designed the system, FPGA devices were optimized to support quick prototyping. FPGAs provide an advantage over custom chips and ASICs, because they can be modified quickly and at low cost after leaving the fabrication vendor.

These FPGA devices are not well suited to realtime reconfiguration, even though it is possible to reconfigure them at run time. The time required to reconfigure an FPGA is a dead time during which no computational progress may be achieved. Hence, minimizing reconfiguration time during computation is a good guideline for effective FPGA use.

Incidentally, most Myricom products contain FPGA devices which are configured once in their life time, before being shipped from Myricom to their users.

Another guideline for effective use of FPGAs is to concurrently perform as many operations as possible.

The use of FPGAs as compute engines allows the hardware to take on a large range of task parameters through reconfiguration.

All three FPGA computing tasks (P1, P3, and P4) correlate a sliding mask with image data (possibly quantized by a threshold). They all need some variation of:

$$G(i,j)=\sum_{u=0}^{31}\sum_{v=0}^{31}F(\ B(u,v),\ M(i+u,j+v)\ )$$

Where G(i,j) is SM(i,j), BS(i,j) or SS(i,j), and B(u,v) is a bit from either mask.

Our design uses 21 units, {U(j), for j=0,...,20}, such that G(i,j) is computed by U(j). Hence, j is spread ***in space*** (concurrently, 21 times) while i, u, and v are spread ***in time*** (sequentially) in 21*32*32 cycles..

This scheme covers the entire 21x21 search area, by computing each search-row (21 pixels wide) in parallel.

A simple example may help in understanding the operation. For simplicity, in this example the image data is of size 6x6 (instead of 52x52) where a and b range over 0...5. The mask size is 3x3 (instead of 32x32) where u and v range over 0...2. The search area is 4x4 (instead of 21x21), because 4=6-3+1 (similar to 21=52-32+1), where i and j range over 0...3.

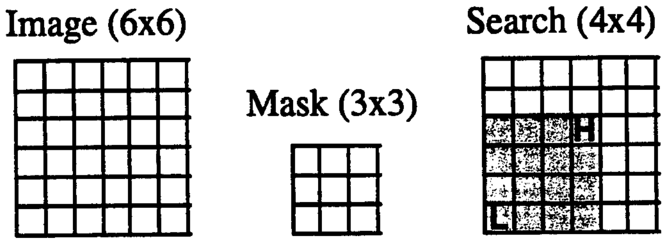


Figure 7: The area sizes for the given example

The P1 computing task is to compute for (i,j=0...3):

```
SM(i,j)= B(0,0)*M(i+0,j+0) + B(0,1)*M(i+0,j+1) + B(0,2)*M(i+0,j+2) +
         B(1,0)*M(i+1,j+0) + B(1,1)*M(i+1,j+1) + B(1,2)*M(i+1,j+2) +
         B(2,0)*M(i+2,j+0) + B(2,1)*M(i+2,j+1) + B(2,2)*M(i+2,j+2)
```

At the first "i-sequence", for i=0, U0 computes SM00, U1 computes SM01, U2 computes SM02, and U3 computes SM03.

Let's write specifically the expressions for SM(0,j), j=0...3, using a shorthand notation:

```
U0: SM00=B00*M00+B01*M01+B02*M02+B10*M10+B11*M11+B12*M12+B20*M20+B21*M21+B22*M22
U1: SM01=B00*M01+B01*M02+B02*M03+B10*M11+B11*M12+B12*M13+B20*M21+B21*M22+B22*M23
U2: SM02=B00*M02+B01*M03+B02*M04+B10*M12+B11*M13+B12*M14+B20*M22+B21*M23+B22*M24
U3: SM03=B00*M03+B01*M04+B02*M05+B10*M13+B11*M14+B12*M15+B20*M23+B21*M24+B22*M25
```

Each of these expressions has 9 terms, each is evaluated and accumulated in 9 successive cycles as shown below:

```
Cycle:    0     1     2     3     4     5     6     7     8
 i:       0     0     0     0     0     0     0     0     0
 u:       0     0     0     1     1     1     2     2     2
 v:       0     1     2     0     1     2     0     1     2
U0: SM00=B00*M00+B01*M01+B02*M02+B10*M10+B11*M11+B12*M12+B20*M20+B21*M21+B22*M22
U1: SM01=B00*M01+B01*M02+B02*M03+B10*M11+B11*M12+B12*M13+B20*M21+B21*M22+B22*M23
U2: SM02=B00*M02+B01*M03+B02*M04+B10*M12+B11*M13+B12*M14+B20*M22+B21*M23+B22*M24
U3: SM03=B00*M03+B01*M04+B02*M05+B10*M13+B11*M14+B12*M15+B20*M23+B21*M24+B22*M25
```

When the above sequence ends, it is repeated for i=1, then i=2, and i=3. v is nested in u, nested in i, but parallel in j. Hence, at the second "sequence", when i=1, U0 computes SM10, U1 computes SM11, U2 computes SM12, and U3 computes SM13.

There are several important things to notice:

First, all the units get the same B(u,v) at the same time. This suggests broadcasting the B(u,v) coefficients to all the Uj units.

Second, the image data that is used by unit U(j) at any cycle is used by the unit U(j-1) at the next cycle (except when v returns to 0). This suggests pipelining the pixels, M, through the Uj units.

When u changes, a new set of image pixels has to be processed. This set is the next image row, except when u returns to 0.

In general, when u changes, the (i+u)th row of pixels is used.

The computing structure for implementing the above is shown in the figure below:

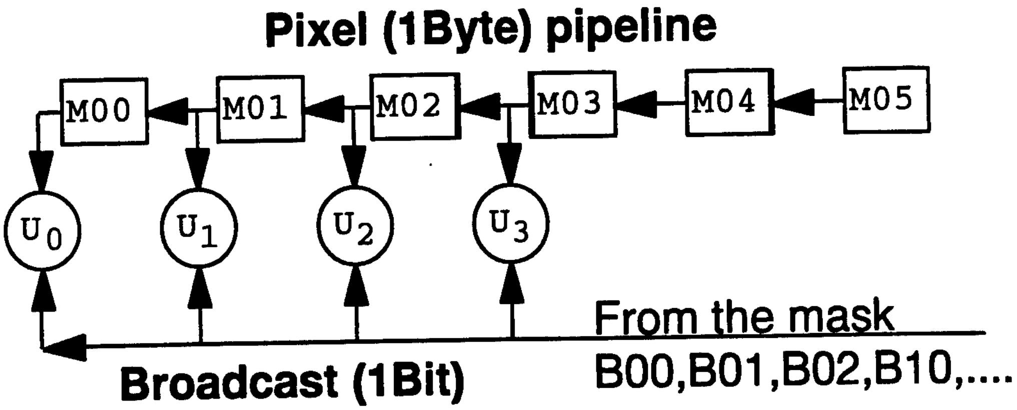


Figure 8: The general structure with pixel pipeline and mask broadcast

In order not to waste time while changing the rows of pixels, the pixels pipeline has the capability either to operate as a pipeline (aka a FIFO) or to be directly loaded from another set of registers.

At every clock cycle each Uj unit performs one operation, v is incremented modulo 3, and the pixel pipeline shifts by one stage (U1 to U0, U2 to U1, ...). When v returns to 0, u is incremented modulo 3, and the pixel pipeline is loaded with the entire (i+u)th row of the image.

When u returns to 0, the results are offloaded from the Uj (using another pipeline that is not shown here), their accumulators are cleared, and i is incremented modulo 4. When i returns to 0, this computing task (P1, P3, or P4) is completed.

The initial loading of the pixel pipeline (with an image row) is from the image-word pipeline that is word wide, hence 4 times faster than the image-pixel pipeline. This speed advantage guarantees that it will be ready with the next image row when u returns to 0.

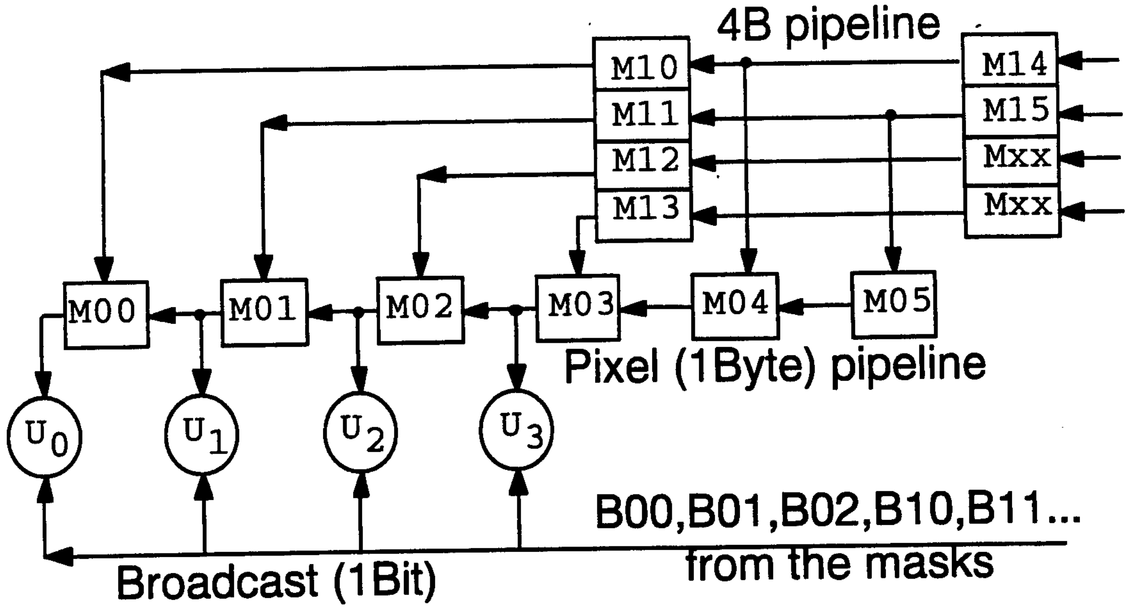


Figure 9: The word-pipeline for initializing the pixel-pipeline

The three computing tasks needed for stages P1, P3, and P4 are:

For P1: Uj has to compute: $SM(i,j) = \sum_{u=0}^{31}\sum_{v=0}^{31} B(u,v)M(i+u, j+v)$

This is accomplished by:

When u returns to 0: Uj = 0
in every other cycle: if (B) then Uj += M
where: B is the bit from the bright mask, being broadcasted,
M is the pixel value available from the pixel pipeline.

The following figure shows the computing structure for P1.

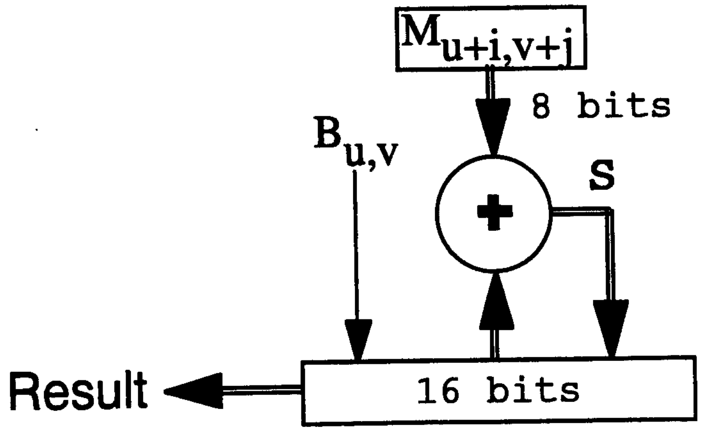


Figure 10: The computing structure for P1

For P3: Uj has to compute $BS(i,j)=\sum_{u=0}^{31}\sum_{v=0}^{31}B(u,v)[\ M(i+u,j+v)\geq TH(i,j)\ ]$:

This is accomplished by:

When u returns to 0: Uj = 0
in every other cycle: if (B) then if (M ≥THj) then Uj++
where B is the bit from the bright mask, being broadcasted,
M is the pixel value available from the pixel pipeline

THj is TH(i,j), as computed by P2.

The following figure shows the computing structure for P3.

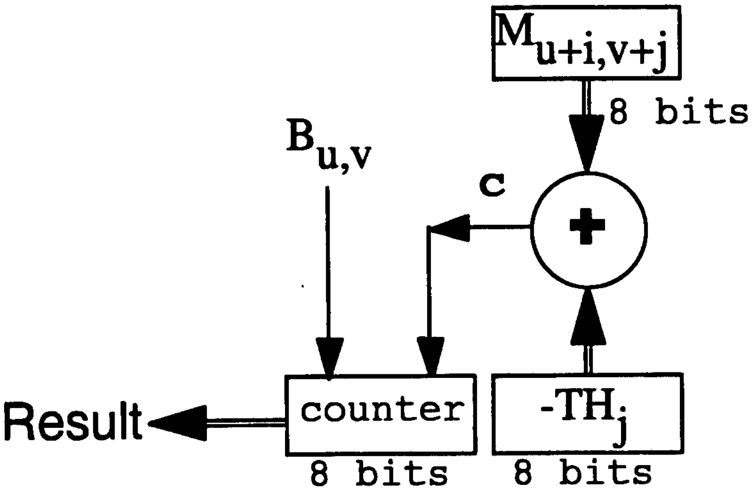


Figure 11: The computing structure for P3

For P4: Uj has to compute: $SS(i,j)=\sum_{u=0}^{31}\sum_{v=0}^{31}S(u,v)[\ M(i+u,j+v)<TH(i,j)\ ]$

This is accomplished by:

When u returns to 0: Uj = 0
in every other cycle: if (B) then if (M<THj) then Uj++
where: B is the bit from the surround mask, being broadcasted,
M is the pixel value available from the pixel pipeline

THj is TH(i,j), as computed by P2.

The following figure shows the computing structure for P4.

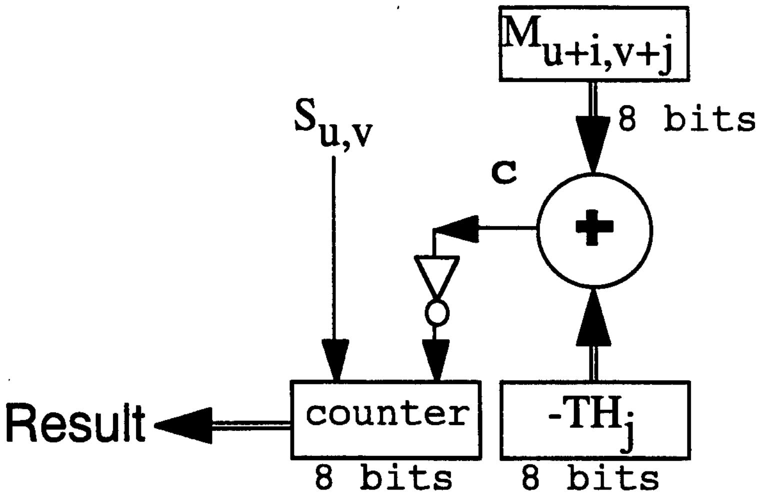


Figure 12: The computing structure for P4

Developing different efficient FPGA programs (or "structures") for P1, P3, and P4 is an interesting approach to solving this problem. At the end of each stage the FPGA device would be reprogrammed (or "reconfigured") with the optimal structure for the next task.

Using this approach matching templates with images at 1KHz requires reconfigurations at 3 KHz, with each reconfiguration limited to 333 µsec (or less if any computing task has to be performed in addition to the reconfigurations). As appealing as this approach may sound, it is not very practical since currently available FPGA devices have typical reconfiguration times of hundreds of milliseconds.

Therefore, we resisted the temptation to perform dynamic reconfigurations. Instead, we designed one structure to perform all three stages.

In all three stages there is a need to bring in pipelined pixels (M), according to the same sequence (v in u in i), to broadcast one bit (B) from a mask, and to modify the value of Uj as a function of a certain pixel (M), and B.

In P1 the modification of Uj is Uj+=M which is an addition of the 8-bit pixel value M, to the 16-bit number Uj. In P3/P4 it's Uj++, where Uj is an 8-bit number.

In P1 the condition for this modification is the value of the B bit. In P3/P4 the condition is, in addition to the B bit, also the comparison of M with THj. Since THj does not depend on u and v, it's a constant until the next change of i.

The modification of Uj is simpler in P3/P4 than in P1, but the evaluation of the condition for modification is simpler in P1 than in P3/P4.

This suggested that the total complexity is about the same in P1 and in P3/P4 and suggested a path to commonality.

An important observation was that adding an 8-bit number to a 16-bit number is the same as adding two 8-bit numbers, M and UjL (L for "low"), and conditionally incrementing UjH (H for "high") if the addition overflows.

This suggests that each stage needs a conditional 8-bit counter, and an 8-bit adder for either adding the pixel M to UjL or for comparing M with THj. This operation is conditional on the mask (B=1).

Both P3 and P4 evaluate exactly the same condition -- comparing M with THj (except that P3 counts if the overflow is 0 and the P4 if it's 1). Therefore they can share a single comparison. This suggests that we perform both P3 and P4 at the same time. We call this combined operation P34.

The FPGA real estate required to perform both P3 and P4 at the same time is less than twice the real estate required for each separately. They share the distribution of M and THj, and the 8-bit adder, but each requires its own broadcast of B (because P3 needs the bright mask whereas P4 needs the surround mask), and each requires its own 8-bit counter.

Hence, the structure of Uj (see following figure) requires an 8-bit adder with the inputs M and R0, whose result is conditionally loaded into R0 and two 8-bit counters, R1 and R2.

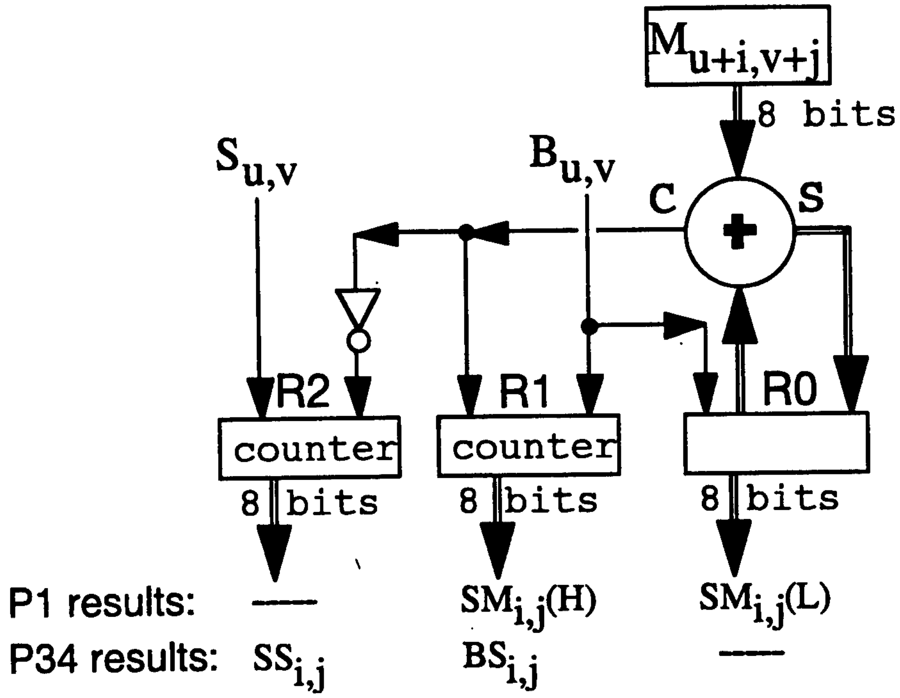

$M_{u+i,v+j}$
8 bits
$S_{u,v}$
$B_{u,v}$
C
+
S
R2
R1
R0
counter
counter
8 bits
8 bits
8 bits
P1 results: —— $SM_{i,j}(H)$ $SM_{i,j}(L)$
P34 results: $SS_{i,j}$ $BS_{i,j}$ ——

Figure 13: The structure for computing both P1 and P34

The use of the 8-bit registers:

Phase:	P1	P34
R0-register:	UjL	-TH(i,j)
R1-counter:	UjH	BS(i,j)
R2-counter:	------	SS(i,j)

Conditions for update:

In P1: increment R1 if (B=1) AND (the 8-bit addition of M to R0 overflows).
In P34: increment R1 if (B=1) AND (the 8-bit addition of M to R0 overflows)
increment R2 if (S=1) AND (the 8-bit addition of M to R0 does not overflow)

The results of P1 are (R0,R1) and of P34 are (R1,R2).

This approach reduces the number of cycles needed to perform P1, P3, and P4 from 3*32*32*21*21 operations to 2*32*32*21 (31.5X improvement).

Above, we described in detail the tasks P1, P3, and P4 that are performed by the FPGA. There are two additional computing tasks, P2 and P5, that are performed by the LANai on the computing node.

P2 computes the threshold, TH(i,j), by dividing the shape sum, SM(i,j), that was computed by the FPGA, by a constant (BC, the number of 1's in the bright mask) and subtracting another constant (the Bias).

The values of Bias, BC, SC, THmax, THmin, BSmin, and SSmin are contained in the templates.

The total number of additions for P1, P3, and P4 is 3*32*32*21*21. P2 uses only 21*21 divide+add operations.

Since the duty cycle of P2 is only 1/(3*32*32) of the entire FPGA operation, our first approach was ***not*** to dedicate any FPGA real estate to this division and to have it performed by the LANai.

After the FPGA completes the task P34, the LANai has to perform task P5, which includes finding the two matches with the highest hit quality, and forwarding the results, according to the architecture that is used (as described in the following section "The data flow through the system").

The hit quality is defined by Q(i,j) = (BS(i,j)/BC + SS(i,j)/SC) / 2 where SC is the number of 1's in the surround mask.

Since typically BC and SC are similar in value (about 100) the LANai on the computing node estimates Q(i,j) by BS(i,j)+SS(i,j) for ordering purposes, and leaves the exact computation of Q(i,j) to the software that runs on a more capable system.

5. Data Flow through the System

The core of the SLD task is to compare an image with a set of templates to find a best correlation, or match. We have seen that the FPGA can effectively perform the computations that are necessary to correlate a single image with a single template. The next question is: how best to partition the task of correlating many images with many templates over an array of FPGA nodes.

In this section we discuss several data flow topologies that may be implemented on top of an arbitrary physical Myrinet topology.

We have implemented three methods for distributing this task over the nodes. The biggest difference between these designs is how messages flow through the system and what these messages consist of. We will refer to these designs as architectures A1, A2 and A3. These are just a few of the myriad topologies that could be used.

5.1 Design A1

In A1, each FPGA node works independently (of the other nodes) to find matches. Each node contains a complete set of templates. The host will give each node a match task (consisting of an image and ranges of templates to correlate the image with), and the node will find the best two matches and return the answer to the host. The host will keep the nodes well supplied with match tasks.

The communications resemble spokes emanating from a central hub (see the figure below) - note that the spokes represent lines of bi-directional communication (match tasks from the host to the nodes, best matches from the nodes to the host). There is no communication between the FPGA nodes, only between the nodes and the host.

This architecture is fault tolerant with respect to node failures - if the host does not hear from a node it can easily resend the match task to another node, and continue to find matches with the remaining nodes. Also, the FPGA nodes can be kept well supplied with tasks - as each node finishes a task it can be given another - resulting in a high utilization of the compute nodes.

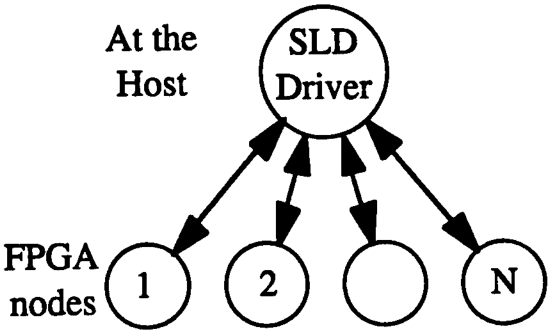


Figure 14: A1, communication is between the host and each node.

In this architecture each node stores all the templates.

5.2 Design A2

A2 partitions the task by distributing the templates over the nodes which are arranged in a chain. Images and their current best matches are passed down the chain from node to node as each node finishes matching an image against the node's subset of the templates. The two best matches so far are passed along with the image and then returned to the host by the last node.

A2 is shown in the following figure, where arrows indicate the direction of data flow. Flow control messages travel in the opposite direction.

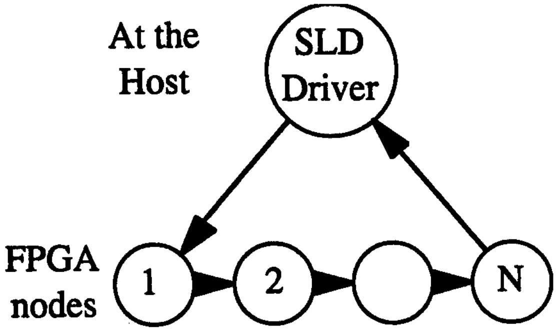


Figure 15: A2, communication along a ring

The data that is being passed is the same throughout the ring; it contains an image, the match specifications (for example, which templates should be correlated with the image) and a description of the two best matches so far. Because match tasks are likely to require that images be compared against consecutive templates, and we wish to balance the computation task across the nodes, we would distribute the templates so that node i out of N nodes has templates (i, i+N, i+2N...).

Each node would store about T/N templates, where N is the number of nodes and T is the total number of templates.

The advantage of A2 over A1 is that it is more scalable with respect to the number of templates. If the template set is very large, each node is A1 may not be able to store all the templates. Also, if the number of nodes in the system is quite large, A2 requires simpler bookkeeping on the host (since all messages come from one node, and the match tasks are naturally queued in the system). However, in A2 if one node or link is lost or becomes faulty, the nodes must re-route the match tasks, redistribute the templates of the "lost" node and then check the image against these templates. Dynamic load balancing is not optimal because, even if the same number of templates have to be matched by each node, the amount of work that each node has to do varies, and a node that is momentarily overwhelmed may create idle nodes downstream.

5.3 Design A3

A3 combines features from A1 and A2 (see figure below). The host may communicate with some rings of nodes (of various sizes) as well as some independent nodes (equivalent to rings of size 1). This hybrid approach would combine the best of both architectures and allow users to customize their system to fit their needs. For example, if the user had different classes of templates (one for matching tanks, another for planes, etc.) then creating a ring of nodes for each class would enable the algorithm to quickly match an image against all templates of one class. Also, using multiple rings would provide better fault tolerance than a single ring - if one node is lost, only one ring (and some percentage of the total nodes) will be affected.

An example of A3 is shown in the following figure, where FPGA nodes 1 through n are in one ring, nodes (n+1) through (N-1) are in another ring, and node N operates independently.

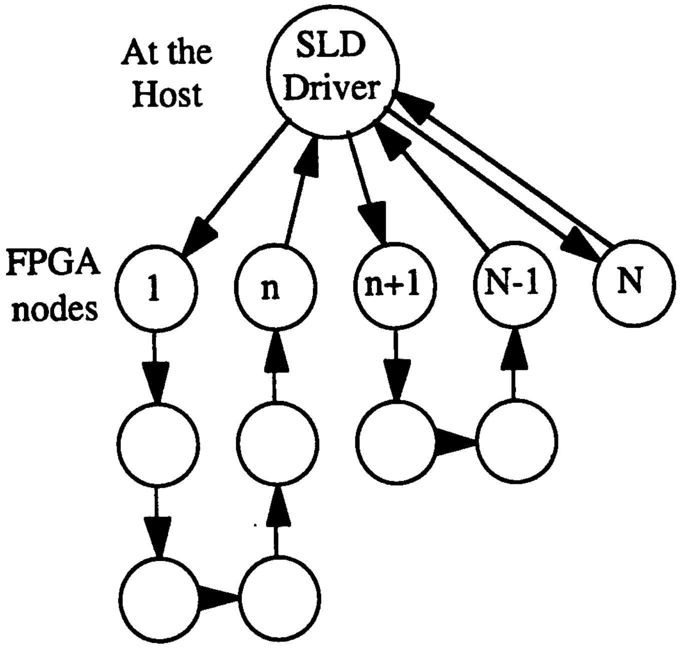


Figure 16: A3, with three rings

Another topology is shown below. It is a variant of A3, with several identical rings. It has 3 rings that are used in parallel, arranged as a 4-stage trellis to increase their fault tolerance, and to improve the dynamic load balancing.

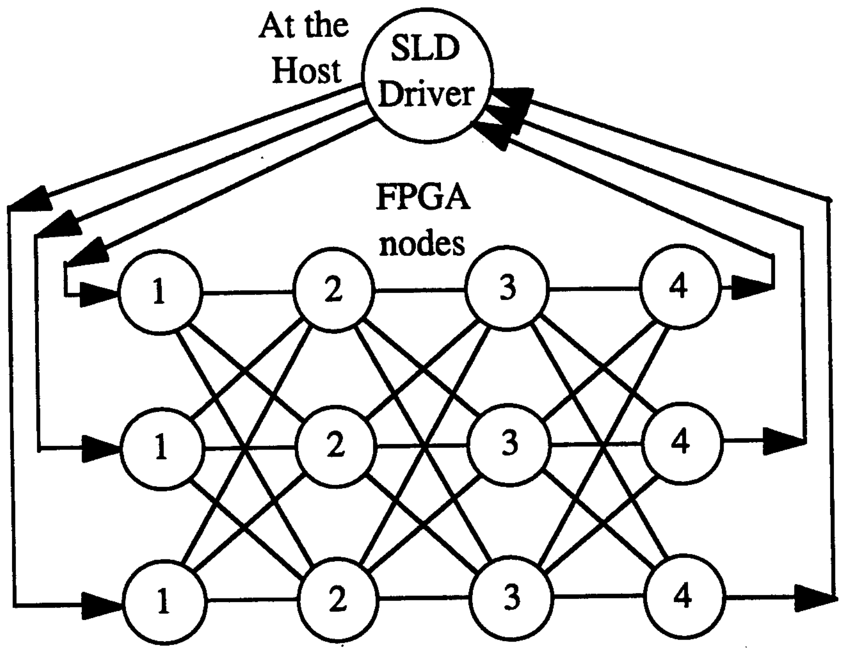


Figure 17: A trellis consisting of 3 parallel identical rings

All nodes in a column, have the same templates set. Upstream flow control messages are used.

5.4 Example

We include an example which illustrates how the optimal number of nodes per ring may be determined. In this example we consider having only one target type, such that all rings must have all the templates.

M match/sec are required. Each node can perform only m match/sec.

T templates are required. Each node can store only t templates.

The number of nodes must be greater than both NP=M/m and NS=T/t ("P" for processing and "S" for storage).

The total number of nodes in the ***system*** must be at least NP.
The total number of nodes in every ***ring*** must be at least NS.

R rings may be used, where R≤int(N/NS) with at least NS nodes in each ring.

For example, let us suppose that there are 23 nodes. 1000 match/sec are needed. Each node can do only 100 match/sec. 200 templates are needed. Each node can store only 40.

NP = 1000/100 = 10, and NS = 200/40 = 5.

N is greater than both NP and NS. (Hence, we can meet the requirements!)

Up to R = 23/5 = 4 rings may be used.

If 4 rings are used, each may have at least 23/4=5 nodes (e.g., 1 ring with 5 nodes and 3 rings with 6 nodes).

If only 3 rings are used, each may have 23/3=7 nodes (e.g., 1 ring with 7 nodes and 2 rings with 8 nodes).

5.5 Summary

In summary, we have presented three data flow designs. In A1, each node works on matches independently. A1 is fault-tolerant, and the FPGA nodes are well-utilized. However, A1 does not scale well to large numbers of templates because each node must store all the templates. A2 places all the nodes in a single communication ring. A2 scales well, but it is less fault tolerant than A1, and nodes may not always have work to do. A3, a hybrid of A1 and A2, consists of placing the nodes in numerous rings. A3 also scales well and is more robust to failures than A2 (but not as robust as A1). In A3, nodes could at time have to wait for work (as in A2).

A multi-stage mesh topology is also presented which illustrates the flexibility of the physical network used in the system.

6. Software Functionality

6.1 Introduction

The software provided by Myricom runs on a host machine and on the FPGA nodes. As shown in the figure below, the Myricom consists of four firmware components (labeled A, B, C and D). These four components run concurrently on the host processor, the host's LANai, the FPGA node's LANai and the node's FPGA, respectively.

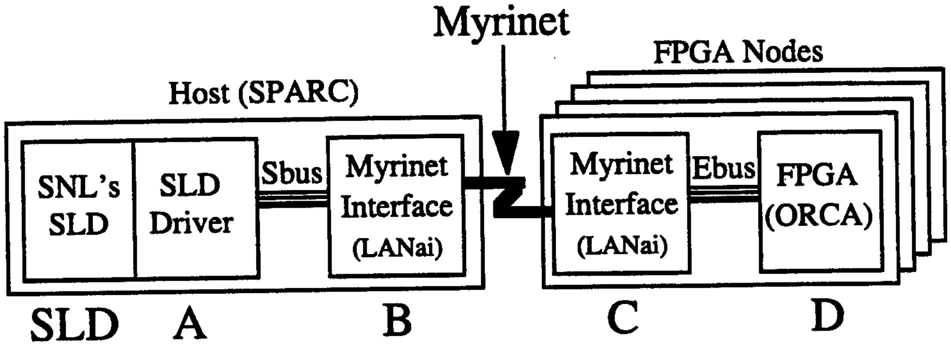


Figure 18: Software configuration (SLD by SNL; A,B,C,D by Myricom)

On the host machine, A resides on the Sparc and communicates with Sandia's SLD software. Component A also communicates over the SBus with B which resides on the host's LANai and is responsible for sending and receiving messages over the Myrinet link.

On the FPGA node, C runs on the LANai, and D runs on the ORCA FPGA.

6.2 Initialization

When an FPGA node is powered up, the ORCA initializes itself, using data from the EEPROM with D* that loads the LANai's SRAM with an initialization program, C*. This program has very limited functionality: it can respond to mapping messages, and it can handle boot messages from a host.

With C* installed, the network can be mapped, and the FPGA nodes can be identified. To map the network, A loads the host's LANai with B*, a mapping program.

6.3 Interfacing with the SLD code

After the hardware powerup of the FPGA nodes, any further processing done by the Myricom firmware is as a result of calls made by the Sandia SLD software to the following functions (which are all in A):

```
fpga_init_node_info();
fpga_load_template();
fpga_send_templates();
fpga_calc_sld();
```

When SLD calls fpga_init_node_info(), it sets off the following set of events. First, the LANai on the host is loaded (by A) with B, which begins execution by sending a boot message to each of the remote nodes. This boot message contains new software for the LANai on the node (C) and may also contain software to configure the FPGA (D). The LANai on the FPGA node (still using C*) replaces itself with C, which then loads the FPGA with D, and then waits for further messages. All FPGA nodes receive identical software (consisting of C and D).

Next, A uses B to send each FPGA node the source route for its "next" node (i.e., where it should send its answers). The choice of the next node depends on the design used (A1, A2, or A3).

After the FPGA nodes have been loaded, SLD calls fpga_load_template() once for each template that it wishes the nodes to remember.

For each template, the software on the host (A) stores the following data provided by SLD:

- bright mask	B(u,v) for u,v=0...31
- surround mask	S(u,v) for u,v=0...31
- template number	
- bias	Bias
- minimum threshold value	THmin
- maximum threshold value	THmax
- minimum bright sum value	BSmin
- minimum surround sum value	SSmin
- bit count of bright mask	BC
- bit count of surround mask	SC

In addition, component A calculates and stores several other values including:

- first non-zero row of either the bright or surround sum	
- bitmap that indicates which rows in the bright mask are all-zero	
- minimum shape sum value	SMmin
- maximum shape sum value	SMmax

Next, SLD calls fpga_send_template() which causes the templates to be sent one at a time to the software running on the nodes. (Depending on the data flow design being used, either all the templates will be sent to all the nodes (under A1), or some subset of the templates will be sent to each node (under A2 or A3)). Now the FPGA nodes are ready to receive match requests.

Finally, SLD will call fpga_calc_sld().When this function is called, SLD provides a pointer to the image, one or two ranges of template configurations to correlate the image with, and a pointer to where the answer should be placed.

As a result of the call to fpga_calc_sld(), a match task will be sent to a node. For A1, the match will be computed at that node and an answer sent back to the host; for rings of nodes as in A2 or A3, the match will continue on from the first node to all the other nodes in the ring and then back to the host.

Once the answer is returned to the host, component A places the best two matches in the location specified by SLD in the fpga_calc_sld() call, and returns.

Note that these software interfaces were designed to closely mirror the interfaces that the SLD code had with existing software.

6.4 Design details

From the perspective of an FPGA node, the difference between A1 and A2 or A3 is very slight. All the nodes are given return routes to use when handing off results (and the nodes don't know nor do they care if this route is to the host or to another node). In A1, each node will be given the complete set of templates whereas in A2 and A3 each node will receive a subset - but to the node this just changes the number of comparisons it must perform.

In A1, the data sent from the host to a node consists of the match task; the data sent from the node to the host contains only the two best matches (i.e., the answer).

In A2 and A3, the data sent from the host to the first node, the data sent between the nodes, and the data sent from the last node to the host are all identical in structure and consist of the match task and the two best answers so far. When the match task comes back to the host, it has passed through all the nodes and the two best answers so far will in fact be the two best matches of the image with all the specified templates. We could use this data structure for A1 as well (note that combining the incoming and outgoing messages for A1 results in the message used for A2 and A3), but in order to keep A1's messages as simple as possible, we chose not to.

In A2 and A3 as in A1 all the nodes are loaded with the same software, and for A2 or A3 the designation of "first" node and "last" node are done by the host, and is completely arbitrary and is also transparent to the nodes.

6.5 LANai Software

The code running on the LANai on the FPGA node, C, has numerous functions. First, it must handle messages received over the Myrinet. This processing is:

```
- if a message has been received
    - if it is a boot message (with new code to run)
        - reboot ourselves and run the new code (C and/or D)
    - if it is a mapping message
        - respond to the message with a mapping-reply message
    - if the message contains data
        - if it has a route to send data to
            - save the route for future use
        - if it has templates
            - store the templates
        - if it has a match task
            - save the image, and the data
            - place it on the task queue
```

Match tasks specify ranges of templates to use. When C receives a match task it places a separate entry for each specified template on a queue for D to process.

A queue entry corresponds to an image-template pair and consists of the following information:

- pointers to: image, bright and surround masks, threshold, and result buffers,
- bitmap that indicates which lines in the bright mask are all-zero,
- the template number (used to access other template information like the minimum bright sum and Bias),
- the image id,
- flags used by the LANai to know when a match task is complete and to aid in freeing buffers that are no longer in use.

Once a match task has been received and the image-template pairs have been placed on a data queue, the LANai's task is to ensure that this data is supplied to the FPGA at the proper time and with the proper pointers for processing to occur.

Ideally, the FPGA should have work to do at all times. Therefore, one of the most important jobs that the LANai on the FPGA node has is to keep the FPGA supplied with work. Recall that there are two tasks that the FPGA must do for each image-template pair. First, it calculates a shape sum (P1) and then it simultaneously calculates the bright and surround sums (P34). There are two additional computations that the LANai must perform. One is to calculate the thresholds from the shape sums (P2) and the other process, P5, is to calculate the two best matches and to report this to the host (or to the next node).

In addition to these computing processes (P2 and P5) the LANai also has a management process, P0, responsible for handling all incoming match tasks.

While the FPGA is working on task P1 or P34, the LANai on the FPGA node can be working on P2 or P5. The software on the LANai supports this by providing four queues for the four tasks (P1, P2, P34, P5).

The diagram below shows the flow of one image-template pair as it passes through the various processes and queues on the FPGA node.

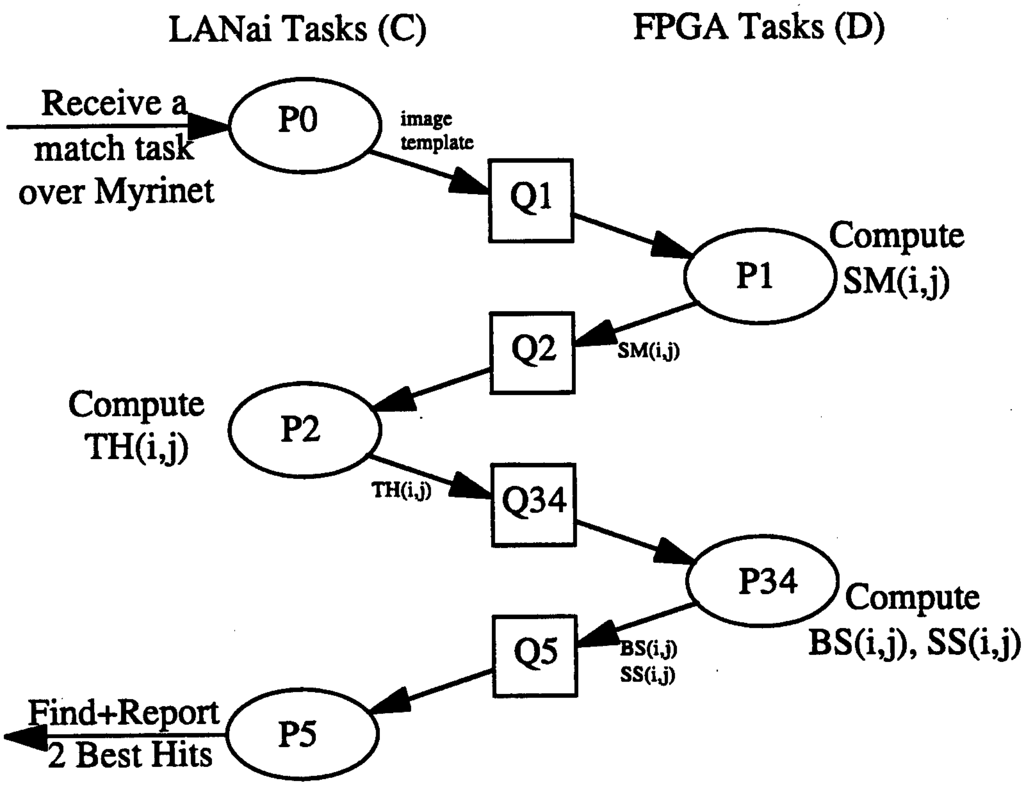


Figure 19: Data flow through the tasks and queues on the FPGA node.

Ideally, the FPGA should be kept well supplied with tasks, and any time that the FPGA is busy with a task (P1 or P34), the LANai should also be busy with a task (P2 or P5). The code on the LANai to support the processing shown in the figure has the form detailed below:

```
- if data on Q34 AND fpga_busy is FALSE
        - dequeue data off Q34
        - send data to fpga to perform P34
        - set fpga_busy to TRUE
        - set current_task to 34
- else if data on Q1 AND fpga_busy is FALSE
        - dequeue data off Q1
        - send data to fpga to perform P1
        - set fpga_busy to TRUE
        - set current_task to 1

- if data on Q5
        - dequeue data off Q5
        - analyze data for two best hits (P5)
        - if this match task is completed (i.e., last template of a set)
                - send answers to host or next node (P5)
- else if data on Q2
        - dequeue data off Q2
        - divide elements from fpga result (P2)
        - queue data on Q34

- if fpga_busy is TRUE
        - if FPGA is done (indicated by a set WAKE bit)
                - set fpga_busy to FALSE
                - if current_task is 1
                                - queue data on Q2
                - if current_task is 34
                                - queue data on Q5
```

By using this design with four queues, the image-template pairs flow easily through the four computational tasks.

The processes are independent and Q34 is checked before Q1, and likewise Q5 before Q2, so that we complete ongoing image-template matches before starting new ones.

Another interesting way to look at the data flow is to see what the LANai and FPGA are doing at any given time (see the table below). Because the FPGA can only perform one task at a time (either P1 or P34) and likewise the LANai can only perform one task at a time (either P2 or P5), any column in the table can have at most 2 entries - one indicating on which image-template pair the FPGA works and one on which pair the LANai works.

time slot:	0	1	2	3	4	5	6	7	8	9
process:										
FPGA P1	0	1			2	3			4	5
LANai P2		0	1			2	3			4
FPGA P34			0	1			2	3		
LANai P5				0	1			2	3	

Figure 20: task and image-template pair versus time slot.

At time slot 4, for example, the FPGA is performing task P1 for image-template pair #2, and the LANai is performing task P5 for pair #1.

The time to perform each of these processes may vary for each template and each template location due to the early-outs employed at each step of the correlation process. The time slots in the table above would be determined by the time taken by the slower of the two processes at work.

Ideally, we would like the FPGA to never have to wait for more data - that is, we would like the combined time to do P2 and P5 (the computations done by the LANai) to be less than the combined time taken by the FPGA to perform tasks P1 and P34 (meaning that the LANai would always be waiting for the FPGA with more data for it to work on).

Currently, the processing on the LANai is a bottleneck and if we could speed up the time taken for P2 or P5 (by moving some of the processing to the FPGA or the host machine), then we could speed up the total processing time. Recall that the process P2 is a division by a constant (BC) and subtraction of another constant (Bias) - a natural job for the FPGA.

7. Hardware

7.1 Structure

The hardware for implementing this system consists of FPGA nodes and baseboards.

The structure of the FPGA nodes is shown below.

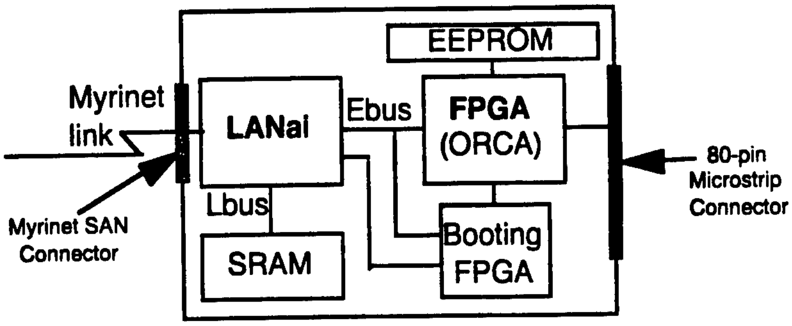


Figure 21: The structure of FPGA nodes

The baseboard, which supports 4 FPGA daughter boards, is a VME-6U board whose structure is shown below.

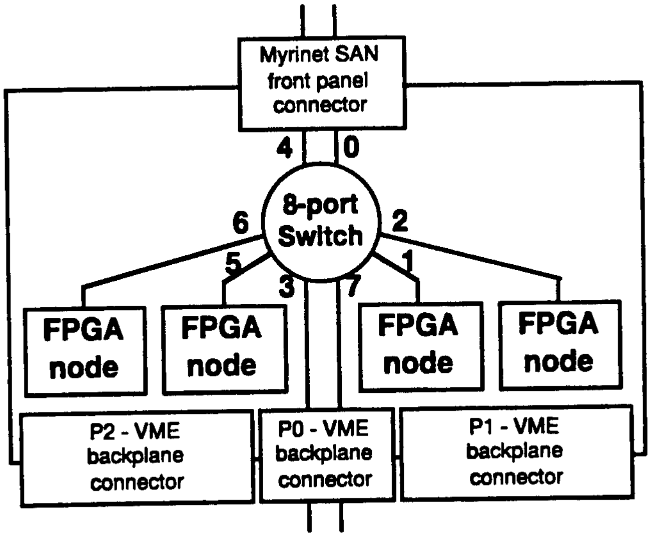

Myrinet SAN front panel connector
4
0
8-port Switch
6
2
5
1
3
7
FPGA node
FPGA node
FPGA node
FPGA node
P2 - VME backplane connector
P0 - VME backplane connector
P1 - VME backplane connector

Figure 22: The structure of the baseboard for 4 FPGA nodes

The following figure shows the actual hardware and the placement of its components. It shows a baseboard, with one (out of possible four) FPGA node plugged into it, and the other side of another node.

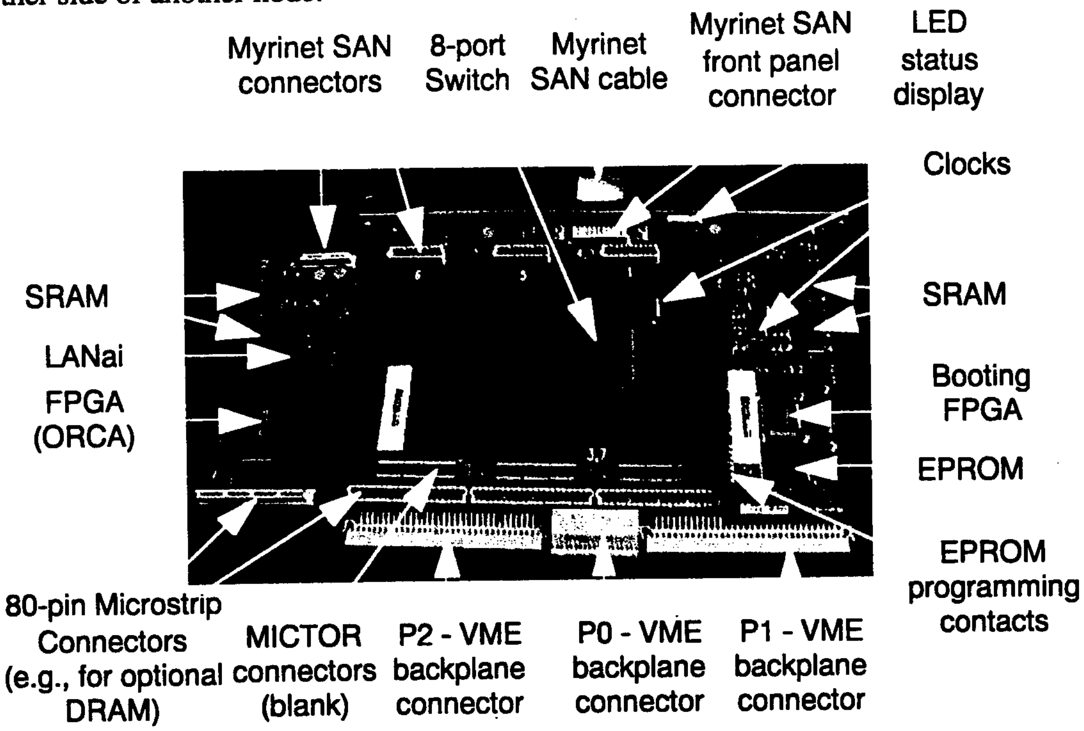

Myrinet SAN connectors
8-port Switch
Myrinet SAN cable
Myrinet SAN front panel connector
LED status display
Clocks
SRAM
LANai
FPGA (ORCA)
SRAM
Booting FPGA
EPROM
EPROM programming contacts
80-pin Microstrip Connectors (e.g., for optional DRAM)
MICTOR connectors (blank)
P2 - VME backplane connector
P0 - VME backplane connector
P1 - VME backplane connector

Figure 23: A picture of the baseboard with two FPGA nodes

This system is a two-level computing system whose first level provides the general purpose infrastructure of network interfacing, message handling, mapping, initialization, etc., by the LANai microprocessor, and second level provides the application-specific computing, which in this case is performed by the ORCA FPGA.

7.2 Initialization

Each of the FPGA nodes could be operated as a self-contained node. At powerup time both the computing FPGA (the ORCA) and the booting FPGA on each node initialize themselves using data from the EEPROM. Then, the ORCA uses data from the EEPROM to load the SRAM of the LANai with its initial program, the C* mentioned in the *initialization* section of the discussion of the *software functionality.*

Once this C* program is loaded, the LANai gets out of its RESET mode, and starts executing that program. From then on, the booting FPGA is waiting for commands from the LANai to reconfigure the computing FPGA, as needed (e.g., with D for ATR).

In our system, this LANai program, C*, depends on programs in the hosts to "program" the FPGA (by providing its configuration data, D) and to replace this C* by the runtime software, C. Loading the FPGA in this manner is done because this is a research and development project, and should be capable of supporting future evolutions as will be required by Sandia.

In a production system, the runtime software could be loaded directly from the EEPROM.

7.3 The Hardware

Each node has:

LANai4.3 RISC processor, 40 MHZ, 3.3V
Lucent ORCA FPGA with 40K gates, 3.3V, speed grade 3
SRAM, 512KB
Booting FPGA
EEPROM

The baseboard (M2M-VME-FB4) has:

Xbar8, 8-port switch
Micro-controller
LED

Four switch ports are connected to the four FPGA nodes (ports 1,2,5,6). Two other ports (ports 0 and 4) are connected to a SAN connector on the front panel, and the remaining two ports (ports 3 and 7) are connected to a P0 connector that is plugged into the VME backplane.

This arrangement allows dedicating a link to each node, accessing all nodes from any single link, or any combination in between. (In the ATR application the link bandwidth is low enough such that using one external link for all 4 nodes does not slow down the operation.)

Each daughter board is connected to the baseboard using two connectors, a Myrinet SAN connector (AMP's 40-signal microstrip connector) and an AMP 80-signal microstrip connector. Having a Myrinet SAN connector allows direct connection of the board to Myrinet SAN cables, should the need arise. The other connector links the ORCA with a slot on the baseboard for a Mictor connector that could be used for connecting additional daughter boards, such as for additional FPGA nodes or DRAM SIMMs.

The only use of the 80-signal connector on our nodes is to communicate between the node's ORCA and a microcontroller on the baseboard, which controls the LED display. This LED, visible on the front panel, is used to indicate the status of the system (such as to show which nodes are running).

The FPGA daughter boards also have contacts for (re)programming their EEPROMs.

A typical VME-6U 19" subrack can house 21 boards. 16 of these may be M2M-VME-FB4 baseboards, connected to 5 M2M-VME-SW10 (or M2M-VME-SW12) as shown below. This arrangement yields 64 FPGA nodes in one 6U subrack.

At the conservative estimate of only 1,500 TSN, i.e., 1500 (template/sec)/node, this arrangement yields 96,000 (template/sec) per 6U-subrack that has only 16 baseboards.

By using the backplane links (over P0) with backplane (or bulkhead) switching, 21 baseboards with 84 FPGA nodes could fit into a single 6U subrack, yielding 126,000 (template/sec) per 6U subrack.

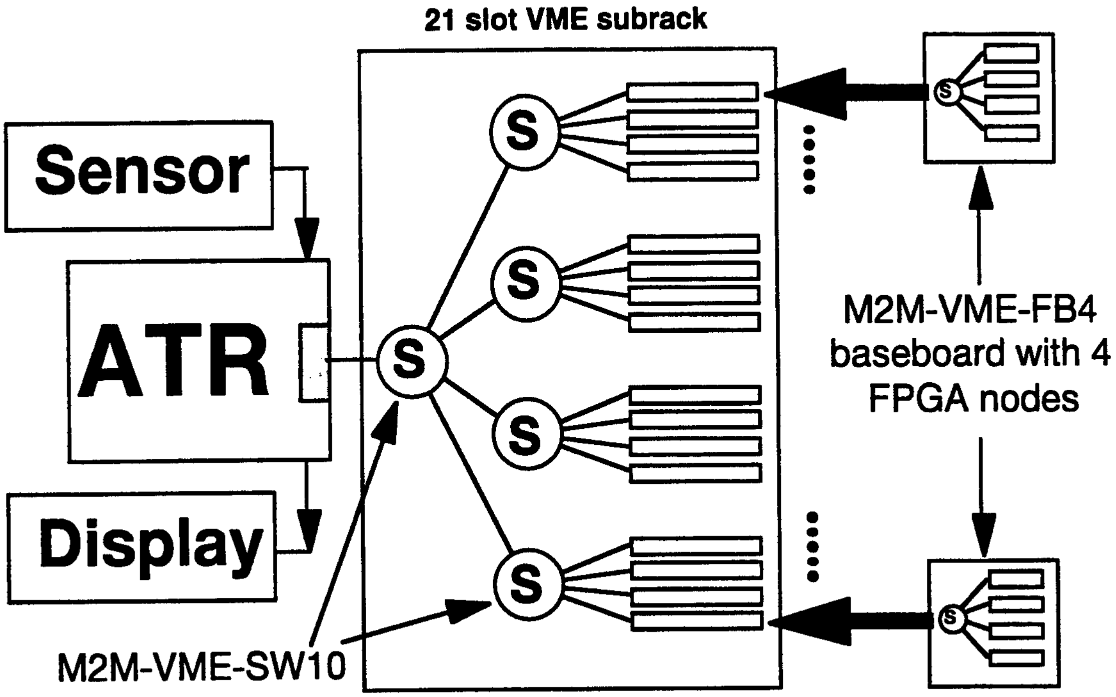


Figure 24: 64 FPGA nodes in a 21 slot VME-6U subrack, using front panel links

If so desired, we can change the form-factor of the FPGA nodes such that 8 of them would fit on a single baseboard that would be redesigned both for handling the new form factor of the nodes and their increased number, either by using two Xbar8 or a single Xbar16.

This would double the number of computing nodes per subrack to 128 or 168, depending on the use of front panel or backplane switches.

8. Performance (Present and Future)

8.1 Test Setup

The test and measurement setup is as shown in the following figure, where

Host Machine:	SparcStation IPX running SunOS 4.1.3
Host Interface:	Myricom M2F-SBus32B board with 512K memory
Switch:	Myricom M2FM-Switch8 switch
FPGA Node:	LANai4.1 running in 3.3-4V at 40Mhz ORCA FPGA 40K operating on 3.3V supply

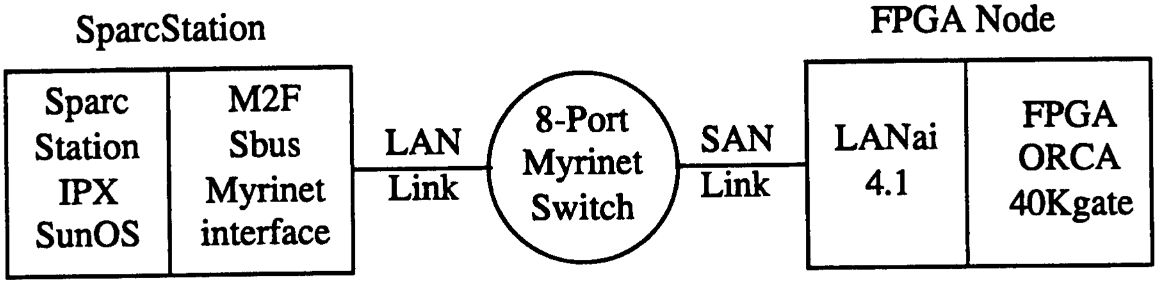


Figure 25: FPGA node test setup

The following measurements were made using the above hardware setup and customized software designed to accurately test the performance of the FPGA nodes, with minimal software overhead.

Notation: the performance of an FPGA node is measured by the number of template it can match in a second, written as TSN for "(templates/sec)/node".

8.2 Performance Without Optimizations (estimated and simulated)

Assuming no optimization, the number of cycles to perform each operation is given below.

P1 (shape sum): 11+8+(32*32+17)*21=	21,880 (cycles/template)/node
P34 (b/s sum) : 11+(8+8+32*32+17)*21 =	22,208 (cycles/template)/node
Total FPGA calculations	44,088 (cycles/template)/node

At 40MHz, a cycle takes 25nsec, hence 44088*25ns=1.102200 msec/template

1/(1.1022 msec/template) = 907 TSN

Without any further optimizations, this FPGA system can deliver ~907 TSN.

This figure was verified by using the ORCA simulator.

8.3 Skipping Zero-Mask-Row (estimated)

The ATR algorithm implemented in the FPGA nodes can take advantage of the sparseness of the templates. While computing P1 and P34 the FPGA algorithm will skip all-zero rows of the template.

The number of mask rows in 72 actual templates for each of 2 targets is 72*2*32=4608. Of these, there are 2206 non-zero bright-rows and 2815 non-zero surround-rows.

The non-zero bright-rows are 2206/4608=16/32 of the total.

The non-zero surround-rows are 2815/4608=20/32 of the total.

Using these ratios the cycle count is reduced to:

P1 (shape sum): 11+8+(32*16+17)*21 = 11128 (cycles/template)/node
P34 (b/s sum) : 11+(8+8+32*20+17)*21 = 16832 (cycles/template)/node
Total for the FPGA computations = 25272 (cycles/template)/node

At 40MHz 25272 cycles/template correspond to ~ 1583 TSN.

This is a 1.75X improvement over the non-optimized performance.

This performance can be achieved only if the FPGA is the pacing process (aka the "bottleneck"), which occurs if the LANai performs its computing tasks (P2 and P5) faster then the FPGA performs its tasks (P1 and P34).

8.4 Skipping Zero-Mask-Row (measured)

This skipping of zero mask rows was implemented in the FPGA firmware, and the performance was measured. 72 templates were matched with one image. This was repeated until 10000 matches were performed.

- When the division of P2 was done by the LANai while the FPGA was processing either P1 or P34, the performance measurements were less than the expected average.

  Average time/template: 0.9725 msec, corresponding to ~ 1028 TSN.

- Next, changes were made to the LANai software in the FPGA node to improve the performance by spreading the 21 divisions (for each search row) between P2 and P5, such that they required more balanced periods.

  Average time/template: 0.76016 msec, corresponding to ~ 1316 TSN.

- Since the division in the LANai is the bottleneck, it should be moved to the FPGA. The entire P2 task (the division and the subtraction of the Bias) should be performed by the FPGA, concurrently with other operations without slowing it. To estimate the resulting performance the following measurement was taken after turning off the division in the LANai (P2), simulating the performance as if the FPGA were performing the division.

  Average time/template: 0.6305 msec, corresponding to ~ 1586 TSN

  This matches the estimated performance of 1583 (templates/sec)/node.

By moving the entire P2 task to the FPGA, the performance should be higher than the measured 1586 TSN (which was achieved by a partial move of P2 to the FPGA).

8.5 Transposing Narrow Masks (future, estimated)

As outlined in Section 3 under Lesson 4, if a template is narrow, it is possible to transpose it and correlate it against a transposed image.

The following is the performance calculation using the same templates (transposed, off line, whenever needed for optimization).

P1 (shape sum): 11+8+(32*12+17)*21 =	8440 (cycles/template)/node
P34 (b/s sum) : 11+(8+8+32*18+17)*21 =	12800 (cycles/template)/node
Total FPGA calculation total =	21240 (cycles/template)/node

Transposing images may be performed by a straight char-char copy: 2704 read cycles and 2704 write cycles = 5408 cycles/image. The host can also send both the original and the transposed images.

Transposing the image has to be done only once for many sets of templates (72 templates for each target).

If the LANai on the FPGA node transposes the image, the cost to transpose should be divided by the number of templates that are to be checked. Therefore, the image overhead is probably on the order of 100 cycles per template.

At 40MHz 21340 cycles/template correspond to ~ 1874 TSN, a 1.14X improvement.

8.6 Skipping Invalid Threshold Lines (future, estimated)

If all the threshold values for an entire search-row are invalid (i.e., each is either below THmin or above THmax) then there is no need to perform P34 on that row since no hit found in that row can be valid.

Our data has:
(2*72 templates)*(16 images)*(21 (search-rows/image)/template)=48384 search-rows.
Of these rows, only 19543 have valid thresholds, or 40%.

With no optimizations we had:

P1 (shape sum): 11+8+(32*32+17)*21 =	21880 (cycles/template)/node
P34 (b/s sum) : 11+(8+8+32*32+17)*21 =	22208 (cycles/template)/node
Total FPGA calculation =	44088 (cycles/template)/node

At 40MHz, 44088 cycles/template correspond to ~ 907 TSN

If invalid-threshold search rows are skipped then we have:

P1 (shape sum): 11+8+(32*32+17)*21 =	21880 (cycles/template)/node
P34 (b/s sum) : 11+(8+8+32*32+17)*21*0.40 =	8880 (cycles/template)/node
Total FPGA calculation =	30760 (cycles/template)/node

At 40MHz, 30760 cycles/template correspond to ~ 1300 TSN, a 1.43X improvement.

8.7 Performance Summary

8.7.1 Present

No optimization (straightforward, brute force) Estimated and verified by simulation.	907 TSN
Skipping zero-rows:	
Measured with the initial P2 and P5	1028 TSN
Measured with the balanced P2 and P5	1316 TSN

8.7.2 Future

Estimated and measured without division by the LANai The division could be implemented by the FPGA	1586 TSN
Transposing masks+images	1.14X
Skipping invalid TH rows	1.43X
Total	2585 TSN

8.8 Future Improvement

The entire P2 task could be moved to the FPGA with no FPGA performance penalty.

Moving the entire P2 task to the FPGA will simplify the entire operation by having the FPGA perform (repeatedly) one task only, P1234, eliminating some of the queues, and significantly simplifying the LANai's coordination/management chores. This will leave the LANai with only the much simpler tasks P0 and P5, as shown below.

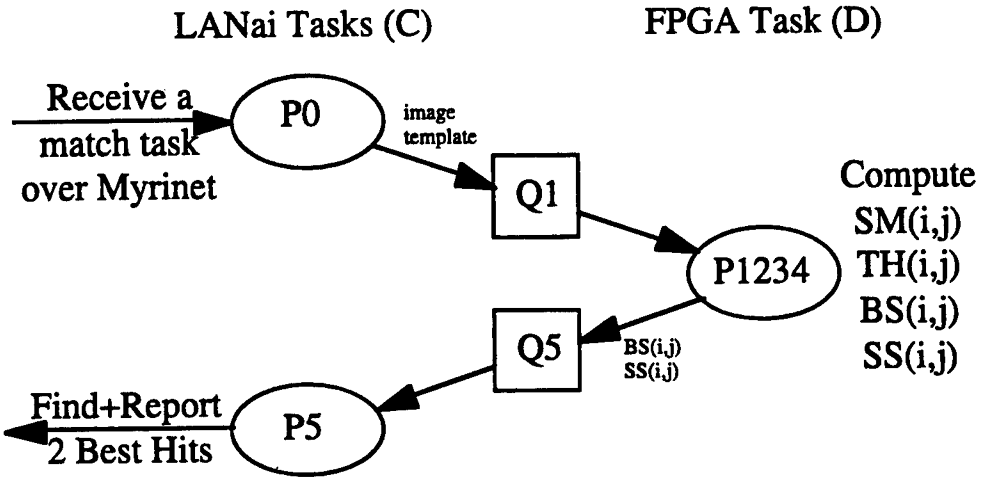


Figure 26: Simplified data flow through the tasks and queues on the FPGA node

If needed, the selection of the 2 best hits could also be moved from the LANai's P5 to the FPGA with no FPGA performance penalty. However, this may not scale easily to higher numbers of best hits.

As long as the FPGA can keep up with the LANai it pays to move tasks from the LANai to the FPGA.

9. Summary

The FPGA node consists of an FPGA (ORCA-40K gate) for computation, a Myrinet network interface (LANai4.3 and SRAM), and a small FPGA with some flash memory for booting. The computation FPGA has no memory directly attached to it; the computation FPGA must access the LANai's memory for template data and image data. The LANai has 512KB of memory attached which is used for network message data, program space, template storage, and image data. A single template requires 270 bytes, and there are 72 templates for each target configuration. The initial implementation supports 6 target configurations per node. Additional target configurations are supported by spreading out the templates amongst the processing nodes.

The correlations are mapped to a linear systolic pipeline. A high degree of parallelism is exploited. In addition to computing an entire row of correlation results in parallel (21 positions), the FPGA performs the address calculations, data loading, and correlations in parallel. Short inter-register paths allow the design to run at 40MHz, which is limited by the clock rate at which external memory can be fetched.

In sequential implementations there are three stages of the computation, thus there is plenty of opportunity for reconfiguration. The first stage consists of the accumulation of 8-bit data into a 16-bit accumulator, the second and third stages consist of comparing 8-bit values and conditionally incrementing counters. In the FPGA implementation, the second and third stage were optimized to be performed in parallel. Instead of reconfiguration for each stage, there is clever design of the processing element so that it can perform all these 3 stages, such that no reconfiguration is necessary, and hence no time is spent on reconfiguration instead of computation.

The compact processing element consists of one 8-bit data input register, one 8-bit accumulator, one 8-bit adder/subtracter, and two 8-bit counters. 21 of these processing elements along with the address generator reside on the FPGA while the divide operation is concurrently done in software in the LANai. There are 1024 (32x32) pixel locations in a mask, and 21 lines in the search area, and two processing steps and some overhead (~1K cycles), for a total of ~44000 (32*32*21*2+1000) cycles, which at 40MHz requires 1.1msec, yielding the rate of ~907 TSN. This was verified by simulations.

Improved performance has been achieved by several additional optimizations. These optimizations are similar to those that microprocessors use: exploiting the sparseness of the templates, and stopping the computation for a given test as soon as a processing stage fails ("early outs"). Since the correlations are in a linear pipeline we cannot exploit the sparseness in a given row, however it is easy to eliminate the calculation for a row if the template is entirely empty for that row.

Skipping zero rows of the template yielded the measured LANai-limited performance of 1316 TSN, which could be raised to 1586 TSN (measured) if some computing tasks were moved from the LANai to the FPGA.

To maximize this possibility, narrow templates could be transposed into short ones and rotated image data could be used for correlation. Another optimization is to abort the computation (an "early out") when an entire row does not pass the shape sum test.

After these optimizations an FPGA node could process 2580 (template/sec)/node, if the data set that we received is indeed representative of the expected data, and if the LANai keeps up with the FPGA nodes.

Four FPGA nodes fit on a single VME-6U baseboard. Using 1316/2580 TSN as the present/future performance yields the performance of 5264/10320 template/sec by a 6U baseboard.

A standard VME-6U subrack has 21 6U boards, which may be all baseboards (with a total of 84 nodes) yielding the performance of 110500/216700 template/sec by the entire subrack.

If this performance is not sufficient, we can design another form-factor for the hardware and populate each baseboard with 8 FPGA nodes, doubling the performance for each baseboard and for the entire subrack.

Populating a subrack with 21 baseboards requires the use of switches outside the subrack. If it is preferred to use only switches that are internal to the subrack without using the P0 backplane, and using only A-links, then only 16 baseboards may be used in a subrack, with 64 FPGA nodes only, yielding the performance of 84200/165100 template/sec by the entire subrack.

Currently the bottleneck of the operation is the division which is still being done in software in the LANai. We could move the entire computing task P2 from the LANai to the FPGA, and would expect approximately 2500 TSN, yielding 210000 match/sec for a VME-6U subrack with 21 baseboards, each with 4 nodes.

The performance could be further increased if one were to reimplement the design with a larger FPGA, even without adding more external memory bandwidth. There is at least a factor of 21 of unexploited parallelism since rows are processed sequentially. Most of the memory fetches are common to the next row of computation, however by computing multiple rows at the same time, the chance that the shape-sum early out will occur decreases. Doubling the hardware on-chip might give close to 2X performance improvement, but 21X hardware increase will give only around 8X performance increase unless memory bandwidth is significantly increased, too.

10. Conclusions

In this report we have described a scalable FPGA system, not just a single accelerator node FPGA. This system is scalable by using an embeddable high-performance networking technology (Myrinet) that has programmable network interfaces. These network interfaces contain microprocessors for local control of the FPGA accelerators. This is an excellent example of two-level multicomputing where the second-level (the FPGA) can contain no control capability, and it is entirely dependent on the first-level (the programmable network interface).

We have demonstrated a scalable FPGA system for an automatic target recognition application. For that application we have measured 1316 TSN and can further improve it to reach 1536 TSN, and possibly further to 2580 TSN, yielding 165120 or 216720 template/sec by using a VME-6U subrack with 16 or 21 baseboards.

This performance scales ***linearly*** with the number of nodes. The only limit to its scalability is the ability of the host to dispatch and handle matching tasks.

We expect further planned design enhancements to increase the computational density advantage by another 33%. Even further performance advantages can be achieved by exploiting the remaining parallelism the design provides by using currently available, larger FPGA devices.

To achieve high performance there were eight main design decisions:

(1) use a proper system architecture (scalable);

(2) have a proper decomposition between hardware and software, not try to execute in FPGAs the complex but not computationally intensive parts of modules that are best left to software in host microprocessors;

(3) use dense packaging to achieve high performance for a given volume, microprocessors typically have a non-trivial amount of support chips;

(4) have short inter-register paths for fast clocking;

(5) have a design that takes advantage of high degree of parallelism;

(6) have a design that can adapt to the dynamic variances in the data to eliminate excess computation, just like in microprocessor software;

(7) do not neglect start and finish overheads; and

(8) minimize reconfiguration since most current FPGA devices reconfigure slowly.

[1v1]

*U.S. GOVERNMENT PRINTING OFFICE: 1999-610-130-81084

DISTRIBUTION LIST

addresses	number of copies
RAYMOND A. LIUZZI AFRL/IFTB 525 BROOKS ROAD ROME, NY 13441-4505	10
MYRICOM, INC. 325 N. SANTA ANITA AVE ARCADIA, CA 91006	5
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	2
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
SOFTWARE ENGR'G INST TECH LIBRARY ATTN: MR DENNIS SMITH CARNEGIE MELLON UNIVERSITY PITTSBURGH PA 15213-3890	1
USC-ISI ATTN: DR ROBERT M. BALZER 4676 ADMIRALTY WAY MARINA DEL REY CA 90292-6695	1
KESTREL INSTITUTE ATTN: DR CORDELL GREEN 1801 PAGE MILL ROAD PALO ALTO CA 94304	1

ROCHESTER INSTITUTE OF TECHNOLOGY 1
ATTN: PROF J. A. LASKY
1 LOMB MEMORIAL DRIVE
P.O. BOX 9887
ROCHESTER NY 14613-5700

AFIT/ENG 1
ATTN:TOM HARTRUM
WPAFB OH 45433-6583

THE MITRE CORPORATION 1
ATTN: MR EDWARD H. BENSLEY
BURLINGTON RD/MAIL STOP A350
BEDFORD MA 01730

UNIV OF ILLINOIS, URBANA-CHAMPAIGN 1
ATTN: ANDREW CHIEN
DEPT OF COMPUTER SCIENCES
1304 W. SPRINGFIELD/240 DIGITAL LAB
URBANA IL 61801

HONEYWELL, INC. 1
ATTN: MR BERT HARRIS
FEDERAL SYSTEMS
7900 WESTPARK DRIVE
MCLEAN VA 22102

SOFTWARE ENGINEERING INSTITUTE 1
ATTN: MR WILLIAM E. HEFLEY
CARNEGIE-MELLON UNIVERSITY
SEI 2218
PITTSBURGH PA 15213-38990

UNIVERSITY OF SOUTHERN CALIFORNIA 1
ATTN: DR. YIGAL ARENS
INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY/SUITE 1001
MARINA DEL REY CA 90292-6695

COLUMBIA UNIV/DEPT COMPUTER SCIENCE 1
ATTN: DR GAIL E. KAISER
450 COMPUTER SCIENCE BLDG
500 WEST 120TH STREET
NEW YORK NY 10027

SOFTWARE PRODUCTIVITY CONSORTIUM 1
ATTN: MR ROBERT LAI
2214 ROCK HILL ROAD
HERNDON VA 22070

AFIT/ENG 1
ATTN: DR GARY B. LAMONT
SCHOOL OF ENGINEERING
DEPT ELECTRICAL & COMPUTER ENGRG
WPAFB OH 45433-6583

NSA/OFC OF RESEARCH 1
ATTN: MS MARY ANNE OVERMAN
9800 SAVAGE ROAD
FT GEORGE G. MEADE MD 20755-6000

AT&T BELL LABORATORIES 1
ATTN: MR PETER G. SELFRIDGE
ROOM 3C-441
600 MOUNTAIN AVE
MURRAY HILL NJ 07974

ODYSSEY RESEARCH ASSOCIATES, INC. 1
ATTN: MS MAUREEN STILLMAN
301A HARRIS B. DATES DRIVE
ITHACA NY 14850-1313

TEXAS INSTRUMENTS INCORPORATED 1
ATTN: DR DAVID L. WELLS
P.O. BOX 655474, MS 238
DALLAS TX 75265

TEXAS A & M UNIVERSITY 1
ATTN: DR PAULA MAYER
KNOWLEDGE BASED SYSTEMS LABORATORY
DEPT OF INDUSTRIAL ENGINEERING
COLLEGE STATION TX 77843

KESTREL DEVELOPMENT CORPORATION 1
ATTN: DR RICHARD JULLIG
3260 HILLVIEW AVENUE
PALO ALTO CA 94304

DARPA/ITO 1
ATTN: DR KIRSTIE BELLMAN
3701 N FAIRFAX DRIVE
ARLINGTON VA 22203-1714

NASA/JOHNSON SPACE CENTER 1
ATTN: CHRIS CULBERT
MAIL CODE PT4
HOUSTON TX 77058

SAIC 1
ATTN: LANCE MILLER
144 WESTFIELD
MCLEAN VA 22102

STERLING IMD INC. 1
KSC OPERATIONS
ATTN: MARK MAGINN
BEECHES TECHNICAL CAMPUS/RT 26 N.
ROME NY 13440

NAVAL POSTGRADUATE SCHOOL 1
ATTN: BALA RAMESH
CODE AS/RS
ADMINISTRATIVE SCIENCES DEPT
MONTEREY CA 93943

HUGHES SPACE & COMMUNICATIONS 1
ATTN: GERRY BARKSDALE
P. O. BOX 92919
BLDG R11 MS M352
LOS ANGELES, CA 90009-2919

SCHLUMBERGER LABORATORY FOR 1
COMPUTER SCIENCE
ATTN: DR. GUILLERMO ARANGO
8311 NORTH FM620
AUSTIN, TX 78720

DECISION SYSTEMS DEPARTMENT 1
ATTN: PROF WALT SCACCHI
SCHOOL OF BUSINESS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CA 90089-1421

SOUTHWEST RESEARCH INSTITUTE 1
ATTN: BRUCE REYNOLDS
6220 CULEBRA ROAD
SAN ANTONIO, TX 78228-0510

NATIONAL INSTITUTE OF STANDARDS 1
AND TECHNOLOGY
ATTN: CHRIS DABROWSKI
ROOM A266, BLDG 225
GAITHSBURG MD 20899

EXPERT SYSTEMS LABORATORY 1
ATTN: STEVEN H. SCHWARTZ
NYNEX SCIENCE & TECHNOLOGY
500 WESTCHESTER AVENUE
WHITE PLAINS NY 20604

NAVAL TRAINING SYSTEMS CENTER 1
ATTN: ROBERT BREAUX/CODE 252
12350 RESEARCH PARKWAY
ORLANDO FL 32826-3224

DR JOHN SALASIN 1
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DR BARRY BOEHM 1
DIR, USC CENTER FOR SW ENGINEERING
COMPUTER SCIENCE DEPT
UNIV OF SOUTHERN CALIFORNIA
LOS ANGELES CA 90089-0781

DR STEVE CROSS 1
CARNEGIE MELLON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH PA 15213-3891

DR MARK MAYBURY 1
MITRE CORPORATION
ADVANCED INFO SYS TECH; G041
BURLINTON ROAD, M/S K-329
BEDFORD MA 01730

ISX 1
ATTN: MR. SCOTT FOUSE
4353 PARK TERRACE DRIVE
WESTLAKE VILLAGE,CA 91361

MR GARY EDWARDS 1
ISX
433 PARK TERRACE DRIVE
WESTLAKE VILLAGE CA 91361

DR ED WALKER 1
BBN SYSTEMS & TECH CORPORATION
10 MOULTON STREET
CAMBRIDGE MA 02238

LEE ERMAN 1
CIMFLEX TEKNOWLEDGE
1810 EMBACADERO ROAD
P.O. BOX 10119
PALO ALTO CA 94303

DR. DAVE GUNNING 1
DARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DAN WELD 1
UNIVERSITY OF WASHINGTON
DEPART OF COMPUTER SCIENCE & ENGIN
BOX 352350
SEATTLE, WA 98195-2350

STEPHEN SODERLAND 1
UNIVERSITY OF WASHINGTON
DEPT OF COMPUTER SCIENCE & ENGIN
BOX 352350
SEATTLE, WA 98195-2350

DR. MICHAEL PITTARELLI 1
COMPUTER SCIENCE DEPART
SUNY INST OF TECH AT UTICA/ROME
P.O. BOX 3050
UTICA, NY 13504-3050

CAPRARO TECHNOLOGIES, INC 1
ATTN: GERARD CAPRARO
311 TURNER ST.
UTICA, NY 13501

USC/ISI 1
ATTN: BOB MCGREGOR
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292

SRI INTERNATIONAL 1
ATTN: ENRIQUE RUSPINI
333 RAVENSWOOD AVE
MENLO PARK, CA 94025

DARTMOUTH COLLEGE 1
ATTN: DANIELA RUS
DEPT OF COMPUTER SCIENCE
11 ROPE FERRY ROAD
HANOVER, NH 03755-3510

UNIVERSITY OF FLORIDA 1
ATTN: ERIC HANSON
CISE DEPT 456 CSE
GAINESVILLE, FL 32611-6120

CARNEGIE MELLON UNIVERSITY 1
ATTN: TOM MITCHELL
COMPUTER SCIENCE DEPARTMENT
PITTSBURGH, PA 15213-3890

CARNEGIE MELLON UNIVERSITY 1
ATTN: MARK CRAVEN
COMPUTER SCIENCE DEPARTMENT
PITTSBURGH, PA 15213-3890

UNIVERSITY OF ROCHESTER 1
ATTN: JAMES ALLEN
DEPARTMENT OF COMPUTER SCIENCE
ROCHESTER, NY 14627

TEXTWISE, LLC 1
ATTN: LIZ LIDDY
2-121 CENTER FOR SCIENCE & TECH
SYRACUSE, NY 13244

WRIGHT STATE UNIVERSITY 1
ATTN: DR. BRUCE BERRA
DEPART OF COMPUTER SCIENCE & ENGIN
DAYTON, OHIO 45435-0001

UNIVERSITY OF FLORIDA 1
ATTN: SHARMA CHAKRAVARTHY
COMPUTER & INFOR SCIENCE DEPART
GAINESVILLE, FL 32622-6125

KESTREL INSTITUTE 1
ATTN: DAVID ESPINOSA
3260 HILLVIEW AVENUE
PALO ALTO, CA 94304

STOLLER-HENKE ASSOCIATES 1
ATTN: T.J. GOAN
2016 BELLE MONTI AVENUE
BELMONT, CA 94002

USC/INFORMATION SCIENCE INSTITUTE 1
ATTN: DR. CARL KESSELMAN
11474 ADMIRALTY WAY, SUITE 1001
MARINA DEL REY, CA 90292

MASSACHUSETTS INSTITUTE OF TECH 1
ATTN: DR. MICHAELE SIEGEL
SLOAN SCHOOL
77 MASSACHUSETTS AVENUE
CAMBRIDGE, MA 02139

USC/INFORMATION SCIENCE INSTITUTE 1
ATTN: DR. WILLIAM SWARTHOUT
11474 ADMIRALTY WAY, SUITE 1001
MARINA DEL REY, CA 90292

STANFORD UNIVERSITY 1
ATTN: DR. GIO WIEDERHOLD
857 SIERRA STREET
STANFORD
SANTA CLARA COUNTY, CA 94305-4125

NCCOSC RDTE DIV D44208 1
ATTN: LEAH WONG
53245 PATTERSON ROAD
SAN DIEGO, CA 92152-7151

SPAWAR SYSTEM CENTER 1
ATTN: LES ANDERSON
271 CATALINA BLVD, CODE 413
SAN DIEGO CA 92151

GEORGE MASON UNIVERSITY 1
ATTN: SUSHIL JAJODIA
ISSE DEPT
FAIRFAX, VA 22030-4444

DIRNSA 1
ATTN: MICHAEL R. WARE
DOD, NSA/CSS (R23)
FT. GEORGE G. MEADE MD 20755-6000

DR. JIM RICHARDSON 1
3660 TECHNOLOGY DRIVE
MINNEAPOLIS, MN 55418

LOUISIANA STATE UNIVERSITY 1
COMPUTER SCIENCE DEPT
ATTN: DR. PETER CHEN
257 COATES HALL
BATON ROUGE, LA 70803

INSTITUTE OF TECH DEPT OF COMP SCI 1
ATTN: DR. JAIDEEP SRIVASTAVA
4-192 EE/CS
200 UNION ST SE
MINNEAPOLIS, MN 55455

GTE/BBN 1
ATTN: MAURICE M. MCNEIL
9655 GRANITE RIDGE DRIVE
SUITE 245
SAN DIEGO, CA 92123

UNIVERSITY OF FLORIDA 1
ATTN: DR. SHARMA CHAKRAVARTHY
E470 CSE BUILDING
GAINESVILLE, FL 32611-6125

AFRL/IFT 1
525 BROOKS ROAD
ROME, NY 13441-4505

AFRL/IFTM 1
525 BROOKS ROAD
ROME, NY 13441-4505

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.